A COMPARATIVE STUDY OF DYNAMIC MAPPING HEURISTICS FOR A CLASS OF INDEPENDENT TASKS ONTO HETEROGENEOUS COMPUTING SYSTEMS

A Thesis
Submitted to the Faculty

of

Purdue University

by

Shoukat Ali

In Partial FulȚllment of the
Requirements for the Degree

of

Master of Science in Electrical Engineering

August 1999

ACKNOWLEDGMENTS

First, I thank my thesis advisor, Professor H. J. Siegel, for his guidance and encouragement throughout my study. His comments and suggestions have been invaluable to me. Without his insight, much of the work in this thesis would not be possible. The support of the members of my advisory committee, Professor Arif Ghafoor and Professor Anthony Maciejewski, is gratefully acknowledged. I thank Professor Debra Hensgen and Professor Taylor Kidd for their valuable comments. I also thank Professor Viktor Prasanna and Dr. Richard Freund for their help in improving some deŢnitions.

Without the support of many of my fellow researchers and colleagues in the Parallel Processing Laboratory many things would have been more difŢcult. I thank Tracy Braun, Surjamukhi Chatterjea, and Mitch Theys for their help in many situations. In particular, I would like to thank Dr. Muthucumaru Maheswaran and Sahra Sedigh for reviewing the manuscript and giving many valuable comments.

This thesis was supported by the DARPA/ITO Quorum Program MSHN Project under NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and under GSA subcontract number GS09K99BH0250. The work in Chapter 5 was also supported by the DARPA/ISO BADD Program and the OfŢce of Naval Research under grant number N00014-97-1-0804. Some of the equipment used in this research was donated by Intel and Microsoft.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Ali, Shoukat, M.S.E.E., Purdue University, August, 1999. A Comparative Study of Dynamic Mapping Heuristics for a Class of Independent Tasks onto Heterogeneous Computing Systems. Major Professor: Howard Jay Siegel.

Heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize their combined performance and/or cost-effectiveness. In an HC system, tasks need to be matched to machines, and the execution of the tasks must be scheduled. This thesis reviews some of the different types of distributed and parallel HC environments, and examines some research issues in HC systems consisting of a network of different machines. The latter purpose is pursued by considering: (1) a quantiŢcation of heterogeneity; (2) a taxonomical examination of a number of mapping (matching and scheduling) heuristics, and (3) a comparative study of a sampling of dynamic mapping heuristics.

A model for heterogeneous environments is developed to allow evaluation of mapping heuristics through simulation. This model allows different degrees and types of heterogeneity to be expressed.

A number of mapping heuristics from the literature are examined with respect to a recently proposed taxonomy for the classiŢcation of mapping heuristics for HC environments. The taxonomy is deŢned in three major parts: (a) the models used for applications and communication requests, (b) the models used for target hardware platforms, and (c) the characteristics of mapping heuristics.

Simulation studies are performed to compare two types of dynamic mapping heuristics: immediate mode and batch mode heuristics. In total, Ţve immediate mode heuristics and three batch mode heuristics are examined. The immediate mode dynamic heuristics consider, to varying degrees and in different ways, task afŢnity for different machines and machine ready times. The batch mode dynamic heuristics consider these factors, as well as the aging of tasks waiting to execute. The simulation results can be used to choose the dynamic mapping heuristic to use in a given heterogeneous environment.

CHAPTER 1

INTRODUCTION

In general, heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize their combined performance and/or cost-effectiveness [15, 21, 36]. HC is an important technique for efŢciently solving collections of computationally intensive problems [18]. As machine architectures become more advanced to obtain higher peak performance, the extent to which a given task can exploit a given architectural feature depends on how well the task's computational requirements match the machine's advanced capabilities. The applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources.

One way of exploiting an HC environment is to decompose a task into subtasks, where each subtask is computationally well suited to a single machine architecture, but different subtasks may have different computational needs (e.g., [55]). These subtasks may share stored or generated data, creating the potential for inter-machine dependencies and data transfer overhead. Once the subtasks are obtained, each subtask is assigned to a machine (matching). Then the subtasks and any inter-machine data transfers are ordered (scheduling) so as to optimize some objective function. The overall problem of matching and scheduling is referred to as mapping. The objective function can be the overall completion time of the task, or a more complex function of multiple requirements.

In some cases, a collection of independent tasks must be mapped, instead of a set of inter-dependent subtasks. Such an independent set of tasks is called a meta-task [20]. An example of meta-task mapping is the mapping of an arbitrary set of independent tasks from different users waiting to execute on a heterogeneous suite of machines. Each task in the meta-task may have associated requirements, such as a deadline and a priority.

One broad objective of the HC community is to design a management system for HC resources (machines, networks, data repositories, etc.) [36]. One important issue within this arena is the design of a mapping system that makes good decisions. Such a system has to be provided with an objective function that it tries to optimize. Current research involves formulating an optimization criterion that will be a function of a set of quality of service (QoS) attributes that are likely to be requested by tasks expected in a given HC environment [31]. This optimization criterion will also serve as a measure of the performance of various mapping approaches available to the community, as well as a means of evaluating overall resource management approaches in general.

The scheduling advisor may have to choose between static and dynamic approaches to the mapping of tasks. Static approaches are likely to sufŢce if the tasks to be mapped are known beforehand, and if predictions about the HC resources are likely to be accurate. Dynamic approaches to mapping are likely to be more helpful if status of the HC system can change randomly, and if the tasks that are supposed to be mapped cannot be determined beforehand. The general problem of developing an optimal matching of tasks to hosts is NP-hard [16]. The goal of this thesis is to: (1) review some of the different types of distributed and parallel HC environments; and (2) examine some research issues in HC systems consisting of a network of different machines. The latter purpose is pursued by considering: (1) a quantiŢcation of heterogeneity; (2) a characterization of techniques for mapping tasks on HC systems; (3) an example HC resource management system; and (4) static and dynamic heuristics for mapping tasks to machines in such HC systems.

Chapters 2 and 3 provide background material for HC systems. Chapter 2 brieţy describes some broad classes of HC systems. An example system for managing resources in HC systems is discussed in Chapter 3. Most of Chapter 4, part of Chapter 5, and all of Chapter 6 present the research performed for this thesis. In Chapter 4, degrees and classes of heterogeneity are described. This chapter charaterizes an HC environment in terms of how “heterogeneous” it is. This characterization is then used to simulate different HC environments, which are needed to evaluate the performance of mapping heuristics under different circumstances. Chapter 5 classiȚes heuristics for mapping independent tasks onto a class of HC systems. This chapter enhances an earlier research effort, and gives sample taxonomical examinations for some heuristics as part of the research for this thesis. Chapter 6 presents simulation studies that compare eight heuristics from two types of dynamic mapping heuristics under a variety of representative HC environments. The comparison of dynamic mapping techniques described in Chapter 6 are the focus, and one of the main results of this thesis. The static mapping methods in Chapter 7 are summarized from [7], and are included in this thesis as related work.

CHAPTER 2

PARALLEL AND DISTRIBUTED HETEROGENEOUS COMPUTING SYSTEMS

There is great variety in the types of parallel and distributed HC systems. In this chapter, the broad Ţeld of HC systems is reviewed as background for the rest of the thesis. Three broad classes are brieţy described: mixed mode, multi-mode, and mixed-machine. The remainder of the thesis focuses on mixed-machine systems.

A mixed-mode HC system refers to a single parallel processing system, whose processors are capable of executing in either the synchronous SIMD or the asynchronous MIMD mode of parallelism, and can switch between the modes at the instruction level with negligible overhead [48]. Thus, a mixed-mode machine is *temporally* heterogeneous, in that it can operate in different modes at different times. This permits different modes of parallelism to be used to execute various portions of a program. The goal of mixed-mode HC systems is to provide in a single machine the best attributes of both the SIMD and the MIMD modes. PASM, TRAC, OPSILA, Triton, and EXECUBE are examples of mixed-mode HC systems that have been prototyped [48].

Because there are various trade-offs between the SIMD and MIMD modes of parallelism, mixed-mode machines can exploit these by matching each portion of a given program with the mode that results in the best overall program performance. Studies have shown that for a given program, a mixed-mode machine may outperform a single-mode machine with the same number of processors (e.g., [17]).

Multi-mode HC is similar to mixed-mode HC in the sense that multiple modes of computation are provided within one machine. However, it is different because all modes of computation can be used simultaneously. An example multi-mode architecture is the image understanding architecture (IUA) [56]. In IUA, heterogeneity is incorporated by having multiple processing layers, where each layer provides a different form and mode of computation. Two levels of MIMD and one level of SIMD processors are included in this system.

Thus, mixed-mode and multi-mode systems represent one extreme of HC, where the heterogeneity resides in a single machine. For more about such systems, see [15].

In mixed-machine HC systems, a heterogeneous suite of machines is interconnected by high-speed links to function as a metacomputer [30] or a grid [18]. (The grid refers to a large-scale pooling of resources to provide dependable and inexpensive access to high-end computational capabilities [18].) A mixed-machine HC system coordinates the execution of various components of a task or meta-task on different machines within the system to exploit the different architectural capabilities available, and achieve increased system performance [36].

CHAPTER 3

MSHN: AN EXAMPLE RESOURCE MANAGEMENT SYSTEM

3.1 Overview

A resource management system (RMS) views the set of heterogeneous machines that it manages as a single virtual machine, and attempts to give the user a location-transparent view of the virtual machine [43]. The RMS should be able to provide the users a higher level of overall performance than would be available from the users' local system alone.

The Management System for Heterogeneous Networks (MSHN, pronounced “mission”) [24] is an RMS for use in HC environments. MSHN is a collaborative research effort that includes the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. It builds on SmartNet, an operational scheduling framework and system for managing resources in an HC environment developed at NRaD [19].

This chapter is a summary of [24]. It has been included to show how the mapping heuristics studied here Ţt into an RMS. Furthermore, most of the research in this thesis has been funded and motivated by the MSHN project.

The technical objective of the MSHN project is to design, prototype, and reŢne a distributed RMS that leverages the heterogeneity of resources and tasks to deliver the requested QoS. To this end, MSHN is investigating: (1) the accurate, task-transparent determination of the end-to-end status of resources; (2) the identiŢcation of different optimization criteria and how non-determinism and the granularity of application and platform models (as outlined by the Purdue HC Taxonomy [6] which is extended in Chapter 5) affect the performance of various mapping heuristics that optimize those criteria; (3) the determination of how security should be incorporated within components as well as how to account for security as a QoS attribute; and (4) the identiŢcation of problems inherent in application and system characterization.

3.2 MSHN Architecture

Figure 3.1 shows the conceptual architecture of MSHN. As seen in the Ţgure, every task running within MSHN makes use of the MSHN Client Library (CL) that intercepts the task's operating system calls.

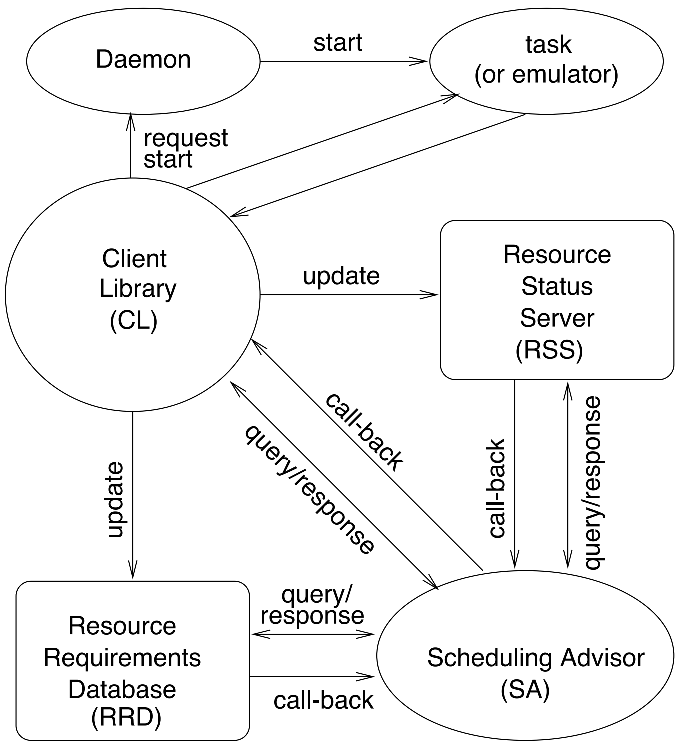

Fig. 3.1. High-level block diagram of the functional architecture of MSHN.

The Scheduling Advisor (SA) determines which set of resources a newly arrived task (or equivalently, a newly started process) should use. Using the terminology from Section 1, the SA is a mapper.

The Resource Status Server (RSS) is a quickly changing repository that maintains information concerning the current availability of resources. Information is stored in the RSS as a result of updates from both the CL and the SA. The CL can update the RSS as to the currently perceived status of resources, which takes into account resource loads due to processes other than those managed by MSHN.

The Resource Requirements Database (RRD) is responsible for maintaining information about the resources that are required to execute a particular task. The RRD's current source of information about a task is the data collected by the CL from the previous runs of the task. The RRD has the ability to maintain very Ţne grain experiential information collected by the CL, and it is hoped that, in the future, it can also be populated with information from smart compilers and directives from task writers.

When the CL intercepts a request to execute a new task, it invokes a mapping request for that task on the SA (assuming that the task requests to be mapped through the SA). The SA queries both the RRD and the RSS. It uses the received information, along with an appropriate search heuristic, to determine the resources that should host the new process. Then, the SA returns the decision to the CL, which, in turn, requests execution of that process through the appropriate MSHN Daemon. The MSHN Daemon invokes the application on its machine.

As a process executes, the CL updates both the RSS and the RRD with the current status of the system resources and the requirements of the process. Meanwhile, the SA establishes call-backs with both the RRD and the RSS to notify the SA if either the status of the resources has signiŢcantly changed, or the actual resource requirements are substantially different than what was initially returned from the RRD. In either case, if it appears that the assigned resources can no longer deliver the required QoS, the application must be terminated or adapted (e.g., use an alternative implementation that may deliver lower QoS, but requires less resources). Upon receipt of a call-back, the SA may require that several of the applications adapt so that more of them can receive their requested QoS.

3.3 Research Issues for MSHN's Scheduling Advisor

The formulation of an optimization criterion for mapping tasks in complex HC environments is currently being researched in the HC community. Resource allocation involves heuristically solving an NP-complete optimization problem. MSHN is developing a criterion that maximizes a weighted sum of values that represents the beneŢts of delivering the required and desired QoS (including security, priorities, and preferences for versions), within the speciŢed deadlines, if any. MSHN attempts to account for both preferences for various versions and priorities. That is, when it is impossible to deliver all of the most preferred QoS within the speciŢed deadlines due to insufŢcient resources, MSHN's optimization criterion is used to decide which resources to allocate to tasks. In MSHN's optimization criterion, deadlines can be simple or complex. That is, sometimes a a piece of information is of signiŢcance to the user only if it is received before a speciŢc time. At other times, a user would like to associate a more general beneŢt function, which would indicate how beneŢcial the information is to the user depending on when it is received. Further information about MSHN's optimization criterion can be found in [31].

The relative performance of mapping heuristics is another research issue. For certain types of HC environments, the MSHN team has obtained a variety of results identifying the regions of heterogeneity where certain heuristics perform better than others for maximizing throughput by minimizing the time at which the last application, of a set of applications, should complete (e.g., [2, 7, 34]). Re-targeting of these heuristics to other optimization criteria is currently underway. Additionally, MSHN team members have performed extensive research into accounting for dependencies among subtasks (e.g., [3, 4, 5, 55]).

The next two chapters present some of the MSHN research in the dynamic and static mapping of meta-tasks in HC environments. The dynamic mapping techniques described in Chapter 6 are one of the main results of this thesis. The static mapping methods in Chapter 7 are summarized from [7], and are included in this thesis as related work.

CHAPTER 4

DEGREES AND CLASSES OF MIXED-MACHINE HETEROGENEITY

4.1 Overview

The goal of this chapter is to characterize an HC environment in terms of how "heterogeneous" it is. This characterization is needed to simulate different HC environments. These simulated HC environments are then used to test the relative performance of different mapping heuristics under different circumstances.

Given a set of heuristics and a characterization of HC environments, one can determine the best heuristic to use in a given environment for optimizing a given objective function. In addition to increasing one's understanding of the operation of different heuristics, this knowledge can help one choose a "good" heuristic in a particular HC environment.

A model for describing an HC system is presented in Section 4.2. Based on that model, two techniques for simulating an HC environment (range-based and coefŢcient-of-variation-based) are described in Section 4.3. The range-based technique is used to create simulated HC environment in Chapters 6 and 7.

4.2 Modeling Heterogeneity

Heuristics that match a task to a machine can vary in the information they use. At the very least, a current candidate task can be assigned to the machine that becomes free soonest (even if the task may take a much longer time to execute on that machine than elsewhere). In another approach, the task may be assigned to the machine where it executes fastest. Or the current candidate task may be assigned to the machine that completes the task soonest, i.e, the machine which minimizes the sum of task execution time and the machine ready time, where machine ready time for a particular machine is the time when that machine becomes free after having executed the tasks previously assigned to it.

The discussion above should reveal that more sophisticated (and possibly wiser) approaches to the mapping problem require estimates of the execution times of all tasks (that can be expected to arrive for service) on all the machines present in the HC suite to make better mapping decisions. The actual task execution times on all machines are not likely to be known. What is typically assumed in the HC literature is that estimates of the expected execution times of tasks on all machines are known (e.g., [22, 29, 49]). These estimates could be built from analytical proŢling of the code and data in the tasks, could be derived from the previous executions of a task on a machine, or provided by the user. (Approaches for doing this estimation based on task proŢling and analytical benchmarking are discussed in [36].)

To better evaluate the behavior of the mapping heuristics, a model of execution times of the tasks on the machines is needed so that the parameters of this model can be changed to investigate the performance of the heuristics under different HC systems and under different sets of tasks to be mapped. One such model consists of an "expected time to compute" (ETC) matrix. The ETC matrix is stored in the mapper, and contains the estimates for the expected execution times of a task on all machines, for all the tasks that are expected to arrive for service. (Although stored in the mapper, the ETC information may be derived from other components of the RMS (e.g, see Chapter 3).) In an ETC matrix, the elements along a row indicate the estimates of the execution times of a given task on different machines, and those along a column give the estimates of execution times of different tasks on a given machine.

The ETC model can be characterized by three parameters: machine heterogeneity, task heterogeneity, and consistency. The variation along a row is referred to as the machine heterogeneity; this is the degree to which the machine execution times vary for a given task [2]. A system's machine heterogeneity is based on a combination of the machine heterogeneities for all tasks (rows). A system comprised mainly of workstations of similar speeds can be said to have “low” machine heterogeneity. A system consisting of diversely capable machines, e.g., a collection of SMP’s, workstations, and supercomputers, may be said to have “high” machine heterogeneity.

Similarly, the variation along a column of an ETC matrix is referred to as the task heterogeneity; this is the degree to which the task execution times vary for a given machine [2]. A system’s task heterogeneity is based on a combination of the task heterogeneities for all machines (columns). “High” task heterogeneity may occur when the computational needs of the tasks vary very much, e.g., when both time-consuming simulations and fast compilations of small programs are performed. “Low” task heterogeneity may typically be seen in the jobs submitted by the users solving similarly complex problems.

Based on the above idea, four categories were proposed for the ETC matrix in [2]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo).

The ETC matrix can be further classiȚed into two classes, consistent and inconsistent [2], which are orthogonal to the previous classiȚcations. For a consistent ETC matrix, if machine m_x has a lower execution time than machine m_y for task t_k, then the same is true for any task t_i. In inconsistent ETC matrices, the relationships among the execution times for different tasks on different machines are unpredictable. The inconsistent case represents a mix of task computational requirements and machine capabilities such that no ordering as that in the consistent case is possible. Inconsistent ETC matrices occur in practice when: (1) there is a variety of different machine architectures in the HC suite (e.g., parallel machines, superscalars, workstations), and (2) there is a variety of different computational needs among the tasks (e.g., readily parallelizable tasks, difȚcult to parallelize tasks, tasks that are țoating point intensive, simple text formatting tasks). Thus, the way in which a task’s needs correspond to a machine’s capabilities may differ for each possible pairing of tasks to machines.

A combination of these two cases, which may be more realistic in many environments, is the semi-consistent ETC matrix, which is an inconsistent matrix with a consistent sub-matrix [7, 34]. As an example, in a given semi-consistent ETC matrix, 50% of the tasks and 25% of the machines may deŢne a consistent sub-matrix.

Formally, semi-consistent is different from both consistent and inconsistent heterogeneities. It does not satisfy the consistency property of the former or the randomness property of the latter.

A trivial case of semi-consistency always exists; for any two machines in the HC suite, *at least* 50% of the tasks will show consistent execution time orderings. For more than two machines, consistency becomes much harder to achieve.

4.3 Generating ETC Matrices

Any method for generating ETC matrices will require that heterogeneity be deŢned mathematically. In the range-based ETC generation technique, the heterogeneity of a set of execution time values is quantiŢed by the range of the execution times [38]. This deŢnition of heterogeneity can be used to generate ETC matrices modeling representative HC environments. The procedures given in this section for generating ETC matrices produce inconsistent ETC matrices. It is shown later in this section how consistent and semi-consistent ETC matrices could be obtained from the inconsistent ETC matrices.

Assume $\underline{m}$ is the total number of machines in the HC suite, and $\underline{t}$ is the total number of tasks expected to be serviced by the HC system. Let $\underline{U(a,\ b)}$ be a number sampled from a uniform distribution with a range from a to b. (Each invocation of $U(a,\ b)$ returns a new sample.) Let $\underline{R_{task}}$ and $\underline{R_{mach}}$ be numbers representing task heterogeneity and machine heterogeneity, respectively, such that higher values for R_{task} and R_{mach} represent higher heterogeneities. Then an ETC matrix $\underline{e}[0..(t-1), 0..(m-1)]$, for a given task heterogeneity and a given machine heterogeneity, can be generated by the range-based method given in Figure 4.1.

As shown in Figure 4.1, each iteration of the outer **for** loop samples a uniform distribution with a range from 1 to R_{task} to generate one value for a vector q. For each element of q thus generated, the m iterations of the inner **for** loop (Line 3) generate one row of the ETC matrix. For the i-th iteration of the outer **for** loop, each iteration of the inner **for** loop produces one element of the ETC matrix by multiplying $q[i]$ with a random number sampled from a uniform distribution ranging from 1 to R_{mach}.

```
(1) for $i$ from 0 to $(t-1)$
(2)     $q[i] = U(1,\ R_{task})$
(3)     for $j$ from 0 to $(m-1)$
(4)         $e[i,j] = q[i] * U(1,\ R_{mach})$
(5)     endfor
(6) endfor
```

Fig. 4.1. The range-based method for generating ETC matrices.

In the range-based ETC generation, it is possible to obtain HiLo ETC matrices with characteristics similar to that of LoHi ETC matrices if $R_{task} = R_{mach}$. In realistic HC systems, the variation that tasks show in their computational needs is generally larger than the variation that machines show in their capabilities. It is assumed here that requirements of high heterogeneity tasks are likely to be more heterogeneous than the capabilities of high heterogeneity machines. Furthermore, low heterogeneity in both machines and tasks is assumed to be same.

Table 4.1 shows typical values for R_{task} and R_{mach} for low and high heterogeneities. Tables 4.2 through 4.5 show four ETC matrices generated by the range-based method. The values of R_{task} and R_{mach} used for generating these ETC matrices are the ones given in Table 4.1.

A variation of the procedure in Figure 4.1 deȚnes the coefȚcient of variation, $\underline{V}$, of execution time values as a measure of heterogeneity (instead of the range of execution time

Table 4.1 Typical values for R_{task} and R_{mach} for a realistic HC system

	High heterogeneity	Low heterogeneity
Task	10^5	10^1
Machine	10^2	10^1

Table 4.2 A HiLo matrix generated by the range-based method using R_t and R_m values of Table 4.1

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	333304	375636	198220	190694	395173	258818	376568
t_2	442658	400648	346423	181600	289558	323546	380792
t_3	75696	103564	438703	129944	67881	194194	425543
t_4	194421	392810	582168	248073	178060	267439	611144
t_5	466164	424736	503137	325183	193326	241520	506642
t_6	665071	687676	578668	919104	795367	390558	758117
t_7	177445	227254	72944	139111	236971	325137	347456
t_8	32584	55086	127709	51743	100393	196190	270979
t_9	311589	568804	148140	583456	209847	108797	270100
t_{10}	314271	113525	448233	201645	274328	248473	170176
t_{11}	272632	268320	264038	140247	110338	29620	69011
t_{12}	489327	393071	225777	71622	243056	445419	213477

values). Let $\underline{\sigma}$ and $\underline{\mu}$ be the standard deviation and mean of a set of execution time values, respectively. Then $V = \sigma/\mu$. The coefŢcient-of-variation-based ETC generation method provides a greater control over spread of values (i.e., heterogeneity) in any given row or column of the ETC matrix than the range-based method.

The coefŢcient-of-variation-based ETC generation method works as follows. A “task vector,” $\underline{q}$, of expected execution times with the desired task heterogeneity must be generated. Essentially, q is a vector whose values represent the task execution times on an “average” machine in the HC suite. For example, if the HC suite consists of an IBM SP/2,

Table 4.3 A HiHi matrix generated by the range-based method using R_t and R_m values of Table 4.1

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	2425808	3478227	719442	2378978	408142	2966676	2890219
t_2	2322703	2175934	228056	3456054	6717002	5122744	3660354
t_3	1254234	3182830	4408801	5347545	4582239	6124228	5343661
t_4	227811	419597	13972	297165	438317	23374	135871
t_5	6477669	5619369	707470	8380933	4693277	8496507	7279100
t_6	1113545	1642662	303302	244439	1280736	541067	792149
t_7	2860617	161413	2814518	2102684	8218122	7493882	2945193
t_8	1744479	623574	1516988	5518507	2023691	3527522	1181276
t_9	6274527	1022174	3303746	7318486	7274181	6957782	2145689
t_{10}	1025604	694016	169297	193669	1009294	1117123	690846
t_{11}	2390362	1552226	2955480	4198336	1641012	3072991	3262071
t_{12}	96699	882914	63054	199175	894968	248324	297691

Table 4.4 A LoLo matrix generated by the range-based method using R_t and R_m values of Table 4.1

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	22	21	6	16	15	24	13
t_2	7	46	5	28	45	43	31
t_3	64	83	45	23	58	50	38
t_4	53	56	26	42	53	9	58
t_5	11	12	14	7	8	3	14
t_6	33	31	46	25	23	39	10
t_7	24	11	17	14	25	35	4
t_8	20	17	23	4	3	18	20
t_9	13	28	14	7	34	6	29
t_{10}	2	5	7	7	6	3	7
t_{11}	16	37	23	22	23	12	44
t_{12}	8	66	47	11	47	55	56

an Alpha server, and a Sun Sparc 5 workstation, then q would represent estimated execution times of the tasks on the Alpha server.

Table 4.5 A LoHi matrix generated by the range-based method using R_t and R_m values of Table 4.1

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
t_1	440	762	319	532	151	652	308
t_2	459	205	457	92	92	379	60
t_3	499	263	92	152	75	18	128
t_4	421	362	347	194	241	481	391
t_5	276	636	136	355	338	324	255
t_6	89	139	37	67	9	53	139
t_7	404	521	54	295	257	208	539
t_8	49	114	279	22	93	39	36
t_9	59	35	184	262	145	287	277
t_{10}	7	235	44	81	330	56	78
t_{11}	716	601	75	689	299	144	457
t_{12}	435	208	256	330	6	394	419

To generate q, two input parameters are needed: μ_{task} and V_{task}. The input parameter, μ_{task} is used to set the average of the values in q. The input parameter V_{task} is the desired coefȚcient of variation of the values in q. The value of V_{task} quantiȚes task heterogeneity, and is larger for higher task heterogeneity. Each element of the task vector q is then used to produce one row of the ETC matrix such that the desired coefȚcient of variation of values in each row is V_{mach}, another input parameter. The value of V_{mach} quantiȚes machine heterogeneity, and is larger for higher machine heterogeneity. Thus μ_{task}, V_{task}, and V_{mach} are the three input parameters for coefȚcient-of-variation-based ETC generation method.

A direct approach to simulating HC environments should use the probability distribution that is empirically found to represent closely the distribution of task execution times. However no benchmarks for HC systems are currently available. Therefore this research uses a distribution which is as ţexible as possible. A gamma distribution can be used for the coefȚcient-of-variation-based ETC generation method because of the ţexibility it allows. The distribution is deȚned in terms of characteristic shape parameter, α, and scale parameter, β. The characteristic parameters of the gamma distribution can be Ţxed to generate different distributions. For example, if α is Ţxed to be an integer, than gamma distribution becomes an Erlang-k distribution. If α is large, then gamma distribution approaches a Gaussian distribution. Future work in this area should attempt to Ţnd out the distribution of task execution times, and see if gamma distribution is a good choice. THe ETC matrices generated here may or may not correspond to actual scenarios. (Note that the uniform distribution can also be used for the coefŢcient-of-variation-based method.)

Figures 4.2 and 4.3 show how a gamma density function changes with the shape parameter α. When the shape parameter increases from two to eight, the shape of the distribution changes from a curve biased to left to a more balanced bell-like curve. Figures 4.4 and 4.5 show the effect on the distribution caused by increase in scale parameter from 16 to 32. The two-fold increase in scale parameter does not change the shape of the graph (the curve is still biased to the left); however the curve now has a twice as large domain (i.e., range on x-axis).

The distribution's characteristic parameters, α and β, can be easily interpreted in terms of μ_{task}, V_{task}, and V_{mach}. For a gamma distribution, $\sigma = \beta\sqrt{\alpha}$, and $\mu = \beta\alpha$, so that $V = 1/\sqrt{\alpha}$. Let $G(\alpha,\ \beta)$ be a number sampled from a gamma distribution with the given parameters. (Each invocation of $G(\alpha,\ \beta)$ returns a new sample.) Figure 4.6 shows the procedure for the coefŢcient-of-variation-based ETC generation.

Given the three input parameters, $V_{task},\ V_{mach}$, and μ_{task}, Line (1) of Figure 4.6 determines the shape parameter α_{task} and scale parameter β_{task} of the gamma distribution

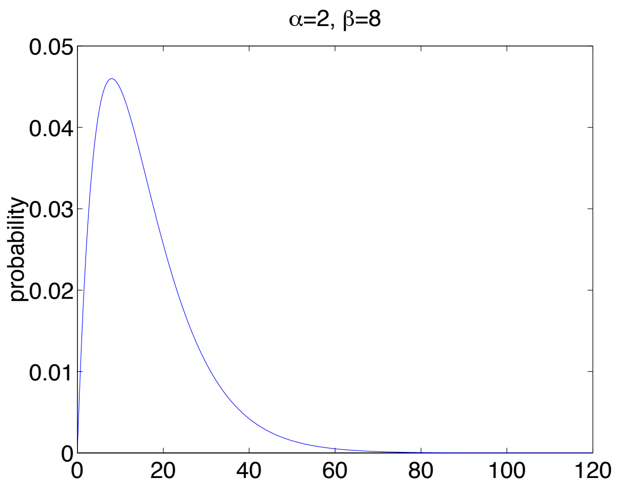

Fig. 4.2. Gamma probability density function for $\alpha = 2,\ \beta = 8$.

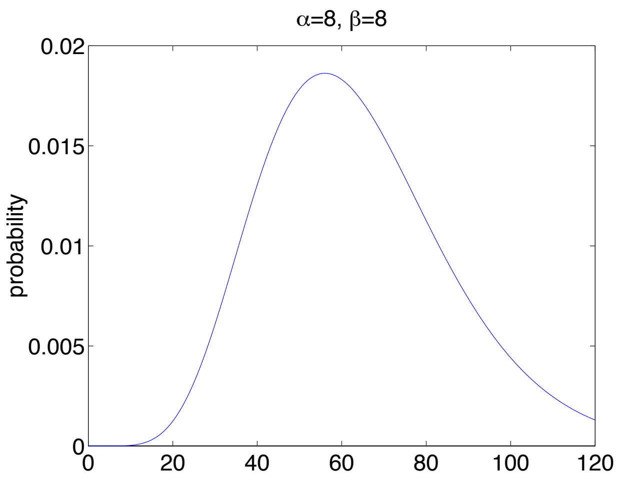

Fig. 4.3. Gamma probability density function for $\alpha = 8,\ \beta = 8$.

which will be later sampled to build the task vector q. Line (1) also calculates the shape parameter α_{mach} to use later in Line (6). In the i-th iteration of the outer **for** loop in Figure 4.6, a gamma distribution with parameters α_{task} and β_{task} is sampled to obtain $q[i]$. Then $q[i]$ is used to determine the scale parameter $\beta_{mach}[i]$ (to be used later in Line (6)). For i-th iteration of the outer **for** loop, each iteration of the inner **for** loop produces one element of the i-th row of the ETC matrix by multiplying $q[i]$ with a random number sampled from a

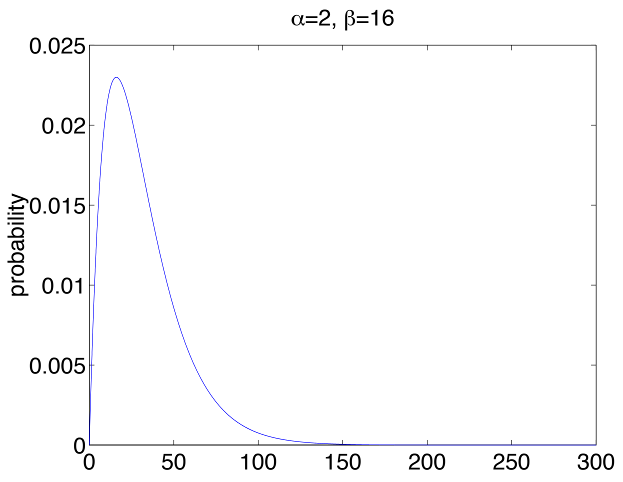

Fig. 4.4. Gamma probability density function for $\alpha = 2,\ \beta = 16$.

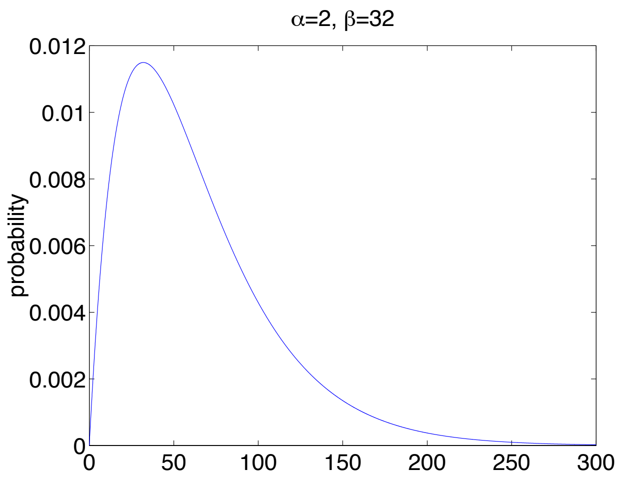

Fig. 4.5. Gamma probability density function for $\alpha = 2,\ \beta = 32$.

gamma distribution with parameters, α_{mach} and $\beta_{mach}[i]$. Note that while each row in the ETC matrix has gamma distributed execution times, the execution times in columns are not gamma distributed.

The ETC generation method of Figure 4.6 can be used to generate HiHi, HiLo, and LoLo ETC matrices, but cannot generate LoHi ETC matrices. To satisfy the heterogeneity

```
(1) $\alpha_{task} = 1/{V_{task}}^2$; $\alpha_{mach} = 1/{V_{mach}}^2$; $\beta_{task} = \mu_{task}/\alpha_{task}$
(2) **for** $i$ from 0 to $(t-1)$
(3)     $q[i] = G(\alpha_{task},\ \beta_{task})$
        /* $q[i]$ will be used as mean of $i$-th row of ETC matrix */
(4)     $\beta_{mach}[i] = q[i]/\alpha_{mach}$     /* scale parameter for $i$-th row */
(5)     **for** $j$ from 0 to $(m-1)$
(6)         $e[i,j] = G(\alpha_{mach},\ \beta_{mach}[i])$     /* $i$-th row */
(7)     **endfor**
(8) **endfor**
```

Fig. 4.6. The coefﬁcient-of-variation-based method for generating ETC matrices.

quadrants of Section 4.2, each column in the ﬁnal ETC matrix should reﬂect the task heterogeneity of the "parent" task vector q. This condition would not necessarily hold if rows of the ETC matrix were produced with a high machine heterogeneity from a task vector of low heterogeneity. This is because a given column may be formed from widely different execution time values from different rows, and may therefore show high heterogeneity as opposed to the intended low heterogeneity. One solution is to take transpose of a HiLo matrix to produce a LoHi one, provided $t = m$. Otherwise, the transposition can be built into the procedure as shown in Figure 4.7.

The procedure in Figure 4.7 is very similar to the one in Figure 4.6. The input parameter μ_{task} is replaced with μ_{mach}. Here, ﬁrst a "machine vector," $\underline{p}$ (with an average value of $\underline{\mu_{mach}}$) is produced. Each element of this "parent" machine vector is then used to generate one low heterogeneity column of the ETC matrix, such that the high machine heterogeneity present in p is reﬂected in all rows.

Tables 4.6 through 4.11 show some sample ETC matrices generated using the coefﬁcient-of-variation-based method. Tables 4.6 and 4.7 both show HiLo ETC matrices. In both tables, the spread of the execution time values in columns is higher than that in rows. ETC matrix in Table 4.7 has a higher task heterogeneity (higher V_{task}) than the ETC

```
(1) $\alpha_{task} = 1/{V_{task}}^2$; $\alpha_{mach} = 1/{V_{mach}}^2$; $\beta_{mach} = \mu_{mach}/\alpha_{mach}$
(2) for $j$ from 0 to $(m-1)$
(3)     $p[j] = G(\alpha_{mach},\ \beta_{mach})$
        /* $p[j]$ will be used as mean of $j$-th column of ETC matrix */
(4)     $\beta_{task}[j] = p[j]/\alpha_{task}$    /* scale parameter for $j$-th column */
(5)     for $i$ from 0 to $(t-1)$
(6)         $e[i,j] = G(\alpha_{task},\ \beta_{task}[j])$    /* $j$-th column */
(7)     endfor
(8) endfor
```

Fig. 4.7. The coefŢcient-of-variation-based method for generating LoHi ETC matrices.

matrix in Table 4.6. This can be seen in a higher spread in the columns of matrix in Table 4.7 than that in Table 4.6.

Tables 4.8 and 4.9 show HiHi and LoLo ETC matrices, respectively. Execution times in Table 4.8 are widely spaced along both rows and columns. Spread of execution times in Table 4.9 is smaller both along columns and rows, because both V_{task} and V_{mach} are smaller.

Tables 4.10 and 4.11 show LoHi ETC matrices. In both tables, the spread of the execution time values in rows is higher than that in columns. ETC matrix in Table 4.11 has a higher machine heterogeneity (higher V_{mach}) than the ETC matrix in Table 4.10. This can be seen in a higher spread in the rows of matrix in Table 4.11 than that in Table 4.10.

The procedures given in Figures 4.1 through 4.7 produce the inconsistent ETC matrices. Consistent ETC matrices can be obtained from the inconsistent ETC matrices generated above by sorting the execution times for all tasks on all machines. From the inconsistent ETC matrices generated above, semi-consistent matrices consisting of an $a \times b$ submatrix could be generated by sorting the execution times across a random subset of b machines for each task in a random subset of a tasks.

It should be noted from Tables 4.10 and 4.11 that the greater the difference in machine and task heterogeneities, the higher the degree of consistency in the inconsistent LoHi ETC

Table 4.6 A HiLo matrix generated by coefŢcient-of-variation-based method. $V_{task} = 0.3,\ V_{mach} = 0.1$

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	628	633	748	558	743	684	740	692	593	554
t_2	688	712	874	743	854	851	701	701	811	864
t_3	965	1029	1087	1020	921	825	1238	934	928	1042
t_4	891	866	912	896	776	993	875	999	919	860
t_5	1844	1507	1353	1436	1677	1691	1508	1646	1789	1251
t_6	1261	1157	1193	1297	1261	1251	1156	1317	1189	1306
t_7	850	928	780	1017	761	900	998	838	797	824
t_8	1042	1291	1169	1562	1277	1431	1236	1092	1274	1305
t_9	1309	1305	1641	1225	1425	1280	1388	1268	1290	1549
t_{10}	881	865	752	893	883	813	892	805	873	915

Table 4.7 A HiLo matrix generated by coefŢcient-of-variation-based method. $V_{task} = 0.5,\ V_{mach} = 0.1$

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	377	476	434	486	457	486	431	417	429	428
t_2	493	370	400	420	502	472	475	440	483	576
t_3	745	646	922	650	791	878	853	791	756	788
t_4	542	490	469	559	488	498	509	431	547	542
t_5	625	666	618	710	624	615	618	599	522	540
t_6	921	785	759	979	865	843	853	870	939	801
t_7	677	767	750	720	797	728	941	717	686	870
t_8	428	418	394	460	434	427	378	427	447	466
t_9	263	289	267	231	243	222	283	257	240	247
t_{10}	1182	1518	1272	1237	1349	1218	1344	1117	1122	1260
t_{11}	1455	1384	1694	1644	1562	1639	1776	1813	1488	1709
t_{12}	3255	2753	3289	3526	2391	2588	3849	3075	3664	3312

matrices. For example, in Table 4.11 all tasks show consistent execution time orderings on all machines except on the machines that correspond to columns 3 and 4. As mentioned

Table 4.8 A HiHi matrix generated by coefﬁcient-of-variation-based method. $V_{task} = 0.6,\ V_{mach} = 0.6$

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	1446	1110	666	883	1663	1458	653	1886	458	1265
t_2	1010	588	682	1255	3665	3455	1293	1747	1173	1638
t_3	1893	2798	1097	465	2413	1184	2119	1955	1316	2686
t_4	1014	1193	275	1010	1023	1282	559	1133	865	2258
t_5	170	444	500	408	790	528	232	303	301	480
t_6	1454	1106	901	793	1346	703	1215	490	537	1592
t_7	579	1041	852	1560	1983	1648	859	683	945	1713
t_8	2980	2114	417	3005	2900	3216	421	2854	1425	1631
t_9	252	519	196	352	958	355	720	168	668	1017
t_{10}	173	235	273	176	110	127	93	276	390	103
t_{11}	115	74	251	71	107	479	153	138	274	189
t_{12}	305	226	860	554	394	344	68	86	223	120

Table 4.9 A LoLo matrix generated by coefﬁcient-of-variation-based method. $V_{task} = 0.1,\ V_{mach} = 0.1$

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	985	1043	945	835	830	1087	1009	891	1066	1075
t_2	963	962	910	918	1078	1091	881	980	1009	981
t_3	782	837	968	960	790	800	947	1007	1115	845
t_4	999	953	892	986	958	1006	1039	1072	1090	1030
t_5	971	972	913	1030	891	873	898	994	1086	1122
t_6	1155	1065	800	1247	980	1103	1228	1062	1011	1005
t_7	1007	1191	964	860	1034	896	1185	932	1035	1019
t_8	1088	864	972	984	736	950	944	994	970	894
t_9	878	967	954	917	942	978	1046	1134	985	1032
t_{10}	1210	1120	1043	1093	1386	1097	1202	1004	1185	1226
t_{11}	910	958	1046	1062	952	1054	1020	1175	850	1060
t_{12}	930	935	908	1155	991	997	828	1062	886	831

Table 4.10 A LoHi matrix generated by coefﬁcient-of-variation-based method. $V_{task} = 0.1,\ V_{mach} = 0.6$

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	1679	876	1332	716	1186	1860	662	833	534	804
t_2	1767	766	1327	711	957	2061	625	626	642	800
t_3	1870	861	1411	932	1065	1562	625	976	556	842
t_4	1861	817	1218	865	1096	1660	587	767	736	822
t_5	1768	850	1465	764	1066	1585	663	863	579	757
t_6	1951	807	1177	914	939	1483	573	961	643	712
t_7	1312	697	1304	921	1005	1639	562	831	633	784
t_8	1665	849	1414	795	1162	1593	577	791	709	774
t_9	1618	753	1283	794	1153	1673	639	787	563	744
t_{10}	1576	964	1373	752	950	1726	699	836	633	764
t_{11}	1693	742	1454	758	961	1781	721	988	641	793
t_{12}	1863	823	1317	890	1137	1812	704	800	479	848

Table 4.11 A LoHi matrix generated by coefﬁcient-of-variation-based method. $V_{task} = 0.1,\ V_{mach} = 2.0$

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
t_1	4784	326	1620	1307	3301	10	103	4449	228	40
t_2	4315	276	1291	1863	3712	11	91	5255	200	47
t_3	6278	269	1493	1181	3186	12	93	4604	235	46
t_4	4945	294	1629	1429	2894	14	87	4724	231	45
t_5	5276	321	1532	1516	2679	12	102	4621	205	46
t_6	4946	293	1467	1609	2661	10	96	3991	255	39
t_7	4802	327	1317	1668	2982	10	90	5090	252	42
t_8	5381	365	1698	1384	3668	12	99	5133	242	38
t_9	5011	255	1491	1386	3061	10	94	3739	216	42
t_{10}	5228	296	1489	1515	3632	12	107	4682	203	38
t_{11}	5367	319	1332	1363	3393	12	72	4769	221	43
t_{12}	4621	258	1473	1501	3124	12	96	4091	199	44

in Section 4.1, these degrees and classes of mixed-machine heterogeneity can be used to characterize HC environments.

CHAPTER 5

CLASSIFYING MAPPING HEURISTICS

5.1 Overview

As mentioned in Chapter 1, in general, Ţnding optimal solutions for the mapping problem and the scheduling of inter-machine communications in HC environments is NP-complete [16], requiring the development of near-optimal heuristic techniques. In recent years, numerous different types of mapping heuristics have been developed (e.g., see [1, 15, 19, 47, 37]). However, selecting a particular heuristic to use in a certain practical scenario remains a difŢcult problem. One of the reasons for this difŢculty is that when one heuristic is presented and evaluated in the literature, typically, different assumptions are made about the underlying target platform than those used for earlier heuristics (e.g., the degree to which the capabilities of machines differ in the HC suite), making comparisons problematic. Similarly, different assumptions about application models complicate comparisons (e.g, the variation among average task execution times). Moreover, the mapping heuristics themselves usually have different characteristics (e.g., different optimization criteria, different execution times). Therefore, a fair comparison of various heuristics is a challenging problem.

These comparison problems are compounded by the fact that there exist no standard set of application benchmarks or target platforms for HC environments. Motivated by these difŢculties, a new taxonomy for classifying mapping heuristics for HC environments is proposed. The Purdue HC Taxonomy is deŢned in three major parts: (1) the models used for applications and communication requests, (2) the models used for target platforms,

This chapter builds on and enhances a conference paper [6].

and (3) the characteristics of mapping heuristics. This new taxonomy builds on previous taxonomies (e.g., [9, 12, 13, 28]).

Before presenting the proposed taxonomy, previous taxonomies from the Ţelds of distributed computing and HC are reviewed below. Then the Purdue HC Taxonomy for mapping heuristics is deŢned. Finally, the beneŢts and possible uses of this new taxonomy are examined.

The research in this chapter was supported partly by the MSHN project and partly by the DARPA/ISO BADD (BattleŢeld Awareness and Data Dissemination) Program. In the BADD Program, communications from a large number of heterogeneous information sources (e.g., databases, sensors) to a large number of heterogeneous destinations (e.g., warŢghters' laptops, proxy servers) must be scheduled over a set of heterogeneous networks [53]. Thus, most of this taxonomy pertains to this communication requests environment also. In general, an application task mentioned in this chapter may correspond to a communication request.

5.2 Previous Taxonomies

Taxonomies related in various degrees to this work have appeared in the literature. In this section, overviews of three related taxonomy studies are given.

A taxonomy classifying scheduling techniques used in general-purpose distributed computing systems is presented in [9]. The classiŢcation of target platforms and application characteristics was outside the scope of that study. The taxonomy in [9] does combine well-deŢned hierarchical characteristics with more general ţat characteristics to differentiate a wide range of scheduling techniques. Several examples of different scheduling techniques from the published literature are also given, with each classiŢed by the taxonomy. In HC systems, however, scheduling is only half of the mapping problem. The matching of tasks to machines also greatly affects execution schedules and system performance. Therefore, the taxonomy proposed in this research also includes categories for platform characteristics and application characteristics, both of which inţuence matching (and scheduling) decisions.

Several different taxonomies are presented in [13]. The Ţrst is the $\underline{EM^3}$ taxonomy, which classiŢes all computer systems into one of four categories, based on execution mode and machine model [12]. The taxonomy proposed here assumes heterogeneous systems from either the $\underline{SEMM}$ (single execution mode, multiple machine models) or the $\underline{MEMM}$ (multiple execution modes, multiple machine models) categories. A "modestly extended" version of the taxonomy from [9] is also presented in [13]. The modiŢed taxonomy introduces new descriptors and is applied to heterogeneous resource allocation techniques. Target platform and application properties were not classiŢed as part of the study (except for considering different parallelism characteristics of applications).

A taxonomy for comparing heterogeneous subtask matching methodologies is included in [28]. The taxonomy focuses on static subtask matching approaches, and classiŢes several speciŢc examples of optimal and sub-optimal techniques. This is a single taxonomy, without the target platform and application parts of the Purdue HC Taxonomy presented in the next section. However, the "optimal-restricted" classiŢcation in [28] includes algorithms that place restrictions on the underlying program and/or multicomputer system.

The Purdue HC Taxonomy uses these studies as a foundation, and extends their concepts to the speciŢc HC mapping problem domain being considered. Relevant ideas from these studies are incorporated into the unique structure of the three-part taxonomy described in the next section, allowing for more detailed classiŢcations of HC mapping heuristics.

5.3 Proposed Taxonomy

It is assumed that a mixed-machine HC system is composed of different machines, with possibly multiple execution models (as in $MEMM$ classiŢcation [13]). The system is deŢned to be heterogeneous if any one or more of the following characteristics varies among machines enough to result in different execution performance among those machines: processor type, processor speed, mode of computation, memory size, number of processors (within parallel machines), inter-processor network (within parallel machines), etc.

The new Purdue HC Taxonomy for describing mapping heuristics for mixed-machine HC systems is deŢned by three major components: (1) application model and communication requests characterization, (2) platform model characterization, and (3) mapping strategy characterization. Earlier taxonomies have focused only on the third item above. To properly analyze and compare mapping heuristics for current and future HC environments, information about both the target platform and the application being executed is needed.

Thus, the Purdue HC Taxonomy classiŢes all three components of an HC environment, and attempts to *qualitatively* deŢne aspects of the environment that can affect mapping decisions and performance. (Doing this *quantitatively* in a thorough, rigorous, complete, and "standard" manner is a long term goal of the HC Ţeld.) This taxonomy is based on existing mapping heuristics found in the literature, as well as previous research and experience within the Ţeld of HC. Each category can, of course, be investigated in more detail. Because research on mapping heuristics is an active and growing Ţeld, it is assumed that this taxonomy will be reŢned and expanded over time to serve as an evolving standard for describing HC mapping heuristics and their assumed environments.

5.3.1 Application Model Characterization

The Ţrst category of the taxonomy deŢnes the *models* used for the applications to be executed on the HC system and for the communications to be scheduled on the inter-machine network. The applications are not classiŢed by functionality, but rather by the traits that deŢne application computational and communication characteristics that may impact mapping decisions and relative mapper performance. Furthermore, the taxonomy includes application traits that may not be realistic, but do correspond to assumptions a given researcher may have made when designing and analyzing mapping heuristics. Typically, such assumptions are made to simplify the mapping problem in some way. For example, many researchers assume a given subtask must receive all of its input data from other subtasks before it can begin executing, when in reality the subtask may be able to begin with only a subset of data. The goal of the taxonomy is to reţect the environment assumed by the mapping heuristic, so that the application model can capture any assumptions made (even if they are unrealistic). The deŢning traits of such an application model are given below.

application type: What type of applications are to be mapped? If all tasks are independent, meta-task mapping is being performed. If there is a single task decomposed into subtasks (recall subtasks have dependencies), it is subtask mapping. One can also have the situation where a meta-task has independent tasks, but some of the tasks have subtasks. In this case, both meta-task and subtask mappings would be necessary. Some of the items mentioned later in this taxonomy apply only to subtask mapping. They will be indicated with the letter “S” written next to them.

communication patterns (S): Does the application have any particular data communication pattern with respect to the source and destination subtasks for each data item to be transferred? Knowledge of data communication pattern can help mapper in reducing communication costs by mapping communicating tasks closer.

data provision and utilization times (S): Can a source subtask release data to the consumers before it completes? Can a consumer subtask begin execution before receiving all of its input data? (As an example, the clustering non-uniform directed graph heuristic in [14] assumes that a subtask cannot send data to other waiting subtasks until it completely Ţnishes executing.) The time at which input data needed by a subtask or output data generated by a subtask can be utilized may vary in relation to subtask start and Ţnish times, and can help mapper overlap the execution of inter-dependent subtasks.

data location: Do tasks require data from special servers? Are data retrieval and storage times considered?

application size: For how many tasks (or subtasks) was the heuristic evaluated? The number of tasks (or subtasks) for which a given heuristic is evaluated can impact the performance of the heuristic for a given metric. For example, it is shown in [34] that a certain class of dynamic heuristics performs better than another class for larger meta-tasks (i.e., ones with a larger number of tasks in it), but not smaller meta-tasks.

temporal distribution: Is the complete set of tasks of a meta-task (or subtasks of a task) to be mapped known *a priori* (static applications), or do the tasks (or subtasks) arrive in a real-time, non-deterministic manner (dynamic applications), or is it a combination of the two?

deadlines: Do the applications have deadlines? This property could be further reȚned into soft and Țrm deadlines, if required. Applications completed by a soft deadline provide the most valuable results. An application that completes after a soft deadline but before a Țrm deadline is still able to provide some useful data. After a Țrm deadline has passed, data from the application is useless.

priorities: Do the applications have priorities? Environments that would require priorities include military systems and machines where time-sharing must be enforced. Priorities are generally assigned by the user (within some allowed range), but the relative weightings given to each priority are usually determined by another party (e.g., a system administrator). Priorities and their relative weightings are required if the mapping strategy is preemptive (deȚned in the mapping strategy characterization).

multiple versions: Do the applications have multiple versions that could be executed? For example, an application that requires an FFT might be able to perform the FFT with either of two different procedures that have different precisions, different execution times, and different resource requirements. What are the relative “values” of the different versions to the user?

QoS requirements: Do certain applications specify any Quality of Service (QoS) requirements? Most QoS requirements, like security, can affect mapping decisions (e.g., not mapping a particular task onto an insecure machine).

interactivity: Are applications user-interactive (i.e., do they depend on real-time user input), and thus must be executed on machines the user has access/clearance for?

task (or subtask) heterogeneity: For each machine in the HC suite, how greatly and with what properties (e.g., probability distribution) do the execution times of the different tasks in the meta-task (or subtasks in the task) vary?

task profiling: Has task proŢling been done? Is the task proŢle available to the mapping heuristic. Task proŢling speciŢes the types of computations present in an application based on the code for the task (or subtask) and the data to be processed [22, 36]. This information may be used by the mapping heuristic, in conjunction with analytical benchmarking (deŢned in the platform model characterization), to estimate task (or subtask) execution time.

execution time representation: How are the estimated execution times of task (or subtasks) modeled? Most mapping techniques require an estimate of the execution time of each task (or subtask) on each machine. The two choices most commonly used for making these estimates from historic or direct information (e.g., that from task proŢling or application writer's advice) are deterministic and distribution modeling. Deterministic modeling uses a Ţxed (or expected) value [55], e.g., the average of ten previous executions of an application. Distribution modeling statistically processes historic knowledge to arrive at a probability distribution for application execution times. This probability distribution is then used to make mapping decisions [33].

5.3.2 Platform Model Characterization

The second category of the taxonomy deŢnes the models used for target platforms available within HC systems. The target platform traits listed are those that may impact mapping decisions, and relative mapper performance. (The target platform is deŢned by the hardware, network properties, and software that constitute the HC suite.) Several existing heuristics make simplifying (but unrealistic) assumptions about their target platforms (e.g., [49] assumes an inŢnite number of machines are available). Therefore, this taxonomy is not limited to a set of realistic target platforms. Instead, a framework for classifying the *models* used for target platforms is provided below. As mentioned in 5.3.1, this is done so that the taxonomy reȚects the environment assumed by a mapping heuristic (even if the environment is unrealistic).

number of machines: Is the number of machines Țnite or inȚnite? Is the number of machines Țxed or variable (e.g., new machines can come on-line)? Furthermore, a given heuristic with a Țnite, Țxed number of machines may treat this number as a parameter that can be changed from one mapping to another.

system control: Does the mapping strategy control and allocate all resources in the environment (dedicated), or are external users also consuming resources (shared)?

task compatibility: Is each machine in the environment able to perform each application, or, for some applications, are special capabilities that are only available on certain machines required? These capabilities could involve issues such as database software, I/O devices, memory space, and security.

machine heterogeneity and architecture: For each task (or subtask), how greatly and with what properties (e.g., probability distribution) do the execution times vary across different machines in the HC suite? For each machine, various architectural features that can impact performance must be considered, e.g., processor type, processor speed, external I/O bandwidth, mode of computation (e.g., SIMD, MIMD, vector), memory size, number of processors (within parallel machines), and inter-processor network (within parallel machines).

code and data access and storage times: How long will it take each machine to access the code and input data it needs to execute a given task? How long will it take each machine to store any resulting output data for a given task? This applies to subtask I/O that is not from/to another subtask, and applies to meta-tasks.

interconnection network (S): What are the various properties of the inter-machine network? Many network characteristics can affect mapping decisions and system performance, including the following: bandwidth, ability to perform concurrent data transfers, latency, switching control, and topology. Most of these network properties are also functions of the source and destination machines. (Volumes of literature already exist on the topic of interconnection networks, therefore, they are not classiȚed here. A general interconnection network taxonomy can be found in [11].)

number of connections (S): How many connections does each machine have to the interconnection network structure or directly to other machines?

concurrent send/receives (S): Can each machine perform concurrent sends and receives of data to other machines (assuming enough network connections)?

overlapped computation/communication (S): Can machines overlap computation and inter-machine communication?

communication time (S): How much time does it take to send data from any one machine to any other? This may be expressed as a function of path establishment time and bandwidth.

analytical benchmarks: Have the machines in the HC suite been evaluated on analytical benchmarks? Are the benchmarking results available to the mapping heuristic? Analytical benchmarking provides a measure of how well each available machine in the HC platform performs on each given type of computation [22, 36]. This information may be used by the mapping heuristic, in conjunction with task proȚling (see the application model characterization), to estimate task (or subtask) execution time.

migration: Do the machines support the migration of tasks (or subtask)? Migration affects the communication patterns among subtasks, and may reduce the advantage of any mapping decision based on pre-migration communication patterns.

5.3.3 Mapping Strategy Characterization

The third category of the Purdue HC Taxonomy deȚnes the characteristics used to describe the mapping strategies. Because the general HC mapping problem is NP-complete, it is assumed that the mapping strategies being classiȚed are only near-optimal techniques.

support for application model: Can the mapping strategy use information about a given trait of the application (as modeled in Section 5.3.1 above)? For example, a mapping strategy that cannot make use of the fact that a given task has multiple versions may be outperformed by the one that does use this information in making mapping decisions.

support for platform model: Can the mapping strategy use information about a given trait of the platform (as modeled in Section 5.3.2 above)? For example, can the mapping strategy take advantage of any support that the platform provides for migration of tasks?

control location: Is the mapping strategy centralized or distributed? Distributed strategies can further be classiȚed as cooperative or non-cooperative (independent) approaches.

execution location: Can a machine within the suite be used to execute the mapping strategy, or is an external machine required?

preemptive: What assumptions does the mapping strategy make about task preemption (e.g., can tasks be stopped and restarted)? Preemptive mapping strategies can interrupt applications that have already begun execution to free resources for more important applications. Applications that were interrupted may be reassigned (i.e., migrated), or may resume execution upon completion of the more important application. Preemptive techniques must be dynamic by deȚnition. Application "importance" must be speciȚed by some priority assignment and weighting scheme, as already discussed in 5.3.1.

fault tolerance: Is fault tolerance considered by the mapping strategy? This may take several forms, such as assigning applications to machines that can perform checkpointing, or executing multiple, redundant copies of an application.

objective function: What is the quantity that the mapping strategy is trying to optimize? This varies widely among strategies, and can make some approaches inappropriate in some situations. The objective function can be as simple as the total execution time for a meta-task, or a more complex function that includes priorities, deadlines, QoS, etc. [31].

application execution time: When making mapping decisions for each machine/task (or subtask) pair, does the mapper use estimated expected execution times or probability distribution's execution time?

dynamic/static: Is the mapping technique dynamic or static? Dynamic mapping techniques operate in real-time (as tasks arrive for immediate execution), and make use of real-time information. Dynamic techniques require inputs from the environment, and may not have a deŢnite end. For example, dynamic techniques may not know the entire set of tasks to be mapped when the technique begins executing; new tasks may arrive at random intervals. Similarly, new machines may be added to the suite. If a dynamic technique has feedback, applications may be reassigned because of the loss of a machine, or application execution taking signiŢcantly longer than expected. In contrast, static mapping techniques take a Ţxed set of applications, a Ţxed set of machines, and a Ţxed set of application and machine attributes as inputs and generate a single, Ţxed mapping. Static mapping techniques have a well-deŢned beginning and end, and each resulting mapping is not modiŢed due to changes in the HC environment or feedback. These techniques are used to plan the execution of a set of tasks for a future time period (e.g., the production tasks to execute on the following day). Some of the items mentioned later in this section apply only to dynamic mapping. They will be indicated with the letter “D” written next to them.

dynamic remapping (D): Does the mapping heuristic require an initial mapping, which it then enhances? For example, a dynamic heuristic with feedback can remap a previous static mapping [37].

on-line/batch (D): Does the dynamic mapping heuristic map a task as soon as it arrives (on-line dynamic mapping) or does it collects arriving tasks into a small batch and then map (batch dynamic mapping)? Batch and on-line dynamic mapping techniques perform differently under different environmental conditions e.g., task arrival rate [34].

data forwarding (S): Is data forwarding considered during mapping [52]? That is, could a subtask executing on a machine receive data from an intermediate machine sooner than from the original source?

duplication (S): Can a given subtask be duplicated and executed on multiple machines to reduce communication overhead?

predictability of mapper execution times: Is the execution time of the mapping strategy predictable? For some heuristics, the execution time of the heuristic can accurately be predicted, e.g., when the mapping heuristic performs a Ţxed, predetermined number of steps with a known amount of computation in each step before arriving at a mapping (e.g., greedy approaches [1]). In contrast, some heuristics are iterative in the sense that the mapping is continually reŢned until some stopping criteria is met, resulting in a number of steps that is not known *a priori*, or in a known number of steps with an unknown amount of work in each step (e.g., genetic algorithms [49, 55]). The execution times of different mapping strategies vary greatly, and are an important property during the comparison or selection of mapping techniques. For example, it is shown in [34] that choice between two mapping heuristics whose performance is comparable may be made based on the heuristics' execution time.

feedback: Does the mapping strategy incorporate real-time feedback from the platform (e.g., machine availability times) or applications (e.g., actual task execution times) into its decisions? Strategies that utilize feedback are dynamic, but not all dynamic strategies have feedback.

5.4 Sample Taxonomical Heuristic Descriptions

This section describes nine heuristics from the literature in terms of the Purdue HC Taxonomy. The heuristics have been examined with respect to each of the three parts of the taxonomy, i.e., the application model, platform model, and mapping strategy characterization.

The k-Percent Best and the Switching Algorithm [34]

Application Model The k-Percent Best and the Switching Algorithm are meta-task mapping heuristics. Each of the two heuristics has been evaluated for a maximum meta-task size of 2000. Temporal distribution of the tasks is not known *a priori*. Tasks have no deadlines, priorities, versions, or QoS speciȚcations. The tasks are not interactive. Task execution times are modeled by sampling a truncated Gaussian distribution whose mean is set to the expected execution time estimate as found from the “expected time to compute” matrix (see Chapter 4). The variance of the Gaussian distribution is set to 300% of its mean [2]. The meta-task heterogeneity is set to an arbitrarily high value; the estimated expected execution times of a given task on the machines in HC system are sampled from a uniform random number distribution with an arbitrarily high range as a measure of high meta-task heterogeneity.

Platform Model Each of the two heuristics is simulated assuming a system of 20 machines. The system is dedicated, i.e., the resources in the system are not being used by external users. Each machine in the system is compatible to all tasks, i.e., each machine can run any task that is submitted to the system. Machine heterogeneity is modeled. No particular interconnection network is assumed. Analytical benchmarking of machines not provided. Tasks are not allowed to migrate to a different machine during execution.

Mapping Heuristic Model Each of the two mapping heuristics is centralized, is located on an external machine, is non-preemptive, does not consider fault tolerance, and has the overall completion time of the meta-task as the objective function. Each of the two mapping heuristics is dynamic, does not require an initial static mapping, maps a task as soon as it arrives, does not remap, does not allow task duplication, has a predictable worst case execution time, and does incorporate feedback from the platform.

The Sufferage and the Max-min [34]

Application Model The Sufferage and the Max-min heuristics are meta-task mapping heuristics. Each of the two heuristics has been evaluated for a maximum meta-task size of 2000. Temporal distribution of the tasks is not known *a priori*. Tasks have no deadlines, priorities, versions, or QoS speciŢcations. The tasks are not interactive. Task execution times are modeled by sampling a truncated Gaussian distribution whose mean is set to the expected execution time estimate as found from the "expected time to compute" matrix (see Chapter 4). The variance of the Gaussian distribution is set to 300% of its mean [2]. The meta-task heterogeneity is set to an arbitrarily high value; the estimated expected execution times of a given task on the machines in HC system are sampled from a uniform random number distribution with an arbitrarily high range as a measure of high meta-task heterogeneity.

Platform Model Each of the two heuristics is simulated assuming a system of 20 machines. The system is dedicated, i.e., the resources in the system are not being used by external users. Each machine in the system is compatible to all tasks, i.e., each machine can run any task that is submitted to the system. Machine heterogeneity is modeled. No particular interconnection network is assumed. Analytical benchmarking of machines not provided. Tasks are not allowed to migrate to a different machine during execution.

Mapping Heuristic Model Each of the two mapping heuristics is centralized, is located on an external machine, is non-preemptive, does not consider fault tolerance, and has the overall completion time of the meta-task as the objective function. Each of the two mapping heuristics is dynamic, does not require an initial static mapping, maps task in batches, remaps tasks, does not allow task duplication, has a predictable worst case execution time, and does incorporate feedback from the platform.

Genetic Algorithm [55]

Application Model The genetic algorithm presented in [55] is a subtask mapping heuristic. Data communication pattern is assumed to have a single source subtask and multiple destination subtasks. Data can not be provided to the consuming subtasks before the source subtask Ţnishes execution. A consumer subtask can begin execution only after receiving all of its input data. Data retrieval and storage times are not considered in this heuristic. The heuristic has been evaluated for a maximum of 100 subtasks. Temporal distribution of the subtasks is known *a priori*. Subtasks have no deadlines, priorities, versions, or QoS speciŢcations. The subtasks are not interactive. The subtask execution times are modeled by sampling a uniform random number distribution with an arbitrary range of 1 to 1000. The subtask heterogeneity is modeled.

Platform Model The genetic algorithm presented in [55] is simulated assuming a system of 20 machines. The system is dedicated, i.e., the resources in the system are not being used by external users. Each machine in the system is compatible to all tasks, i.e., each machine can run any task that is submitted to the system. Machine heterogeneity is modeled. The study assumes a communication system modeled after a HiPPI LAN with a central crossbar switch. Each machine has two links to the central crossbar switch. Concurrent send and receive of data assumed. Communication time between two machines is given by the data item length divided by bandwidth of the link. Analytical benchmarking of machines not provided. Subtasks are not allowed to migrate to a different machine during execution.

Mapping Heuristic Model The mapping heuristic is centralized, is located on an external machine, is non-preemptive, does not consider fault tolerance, and has the overall completion time of the meta-task as the objective function. The mapper does not handle the execution time modeling. The mapping heuristic is static, allows data forwarding, does not allow subtask duplication, does not have a predictable worst case execution time, and does not incorporate feedback from the platform.

A-Schedule, B-Schedule, C-Schedule, and D-Schedule [25]

Application Model A-Schedule, B-Schedule, C-Schedule, and D-Schedule are all meta-task mapping heuristics. Data retrieval and storage times are not considered in these heuristics. The heuristics have not been evaluated for any particular number of tasks. No simulations were performed. Temporal distribution of the tasks is known *a priori*. Tasks have no deadlines, priorities, versions, or QoS speciȚcations. The tasks are not interactive. The task heterogeneity is modeled.

Platform Model The system is dedicated, i.e., the resources in the system are not being used by external users. Each machine in the system is compatible to all tasks, i.e., each machine can run any task that is submitted to the system. Machine heterogeneity is modeled. Analytical benchmarking of machines not provided. Tasks are not allowed to migrate to a different machine during execution.

Mapping Heuristic Model The mapping heuristic is centralized, is located on an external machine, is non-preemptive, does not consider fault tolerance, and has the overall completion time of the meta-task as the objective function. The mapping heuristic is static, does have a predictable worst case execution time, and does not incorporate feedback from the platform.

5.5 Summary

The Purdue HC Taxonomy can be beneȚcial to researchers in several ways. It can allow more meaningful comparisons among different mapping approaches. It can help extend existing mapping work, and recognize important areas of research by facilitating understanding of the relationships that exist among previous efforts. The three-part classiȚcation system provided allows HC researchers to describe mapping heuristics more thoroughly, and to see design and environment alternatives that they might not have otherwise considered during the development of new heuristics. A researcher can also use the taxonomy to Țnd the mapping heuristics that use similar target platform and application models. The mapping heuristics found for similar models can then possibly be adapted or developed further to better solve the mapping problem that is being considered. The taxonomy can also be used to specify the requirements and capabilities of a resource management system, such as MSHN. In the future, this taxonomy could focus research towards the development of a standard set of benchmarks for HC environments. It is expected, as research progresses, that the Purdue HC Taxonomy will be an evolving standard, that is reȚned and extended to incorporate new ideas and Țndings.

CHAPTER 6

DYNAMIC HEURISTICS FOR MAPPING META-TASKS IN HC SYSTEMS

6.1 Introduction

HC is an important technique for efŢciently solving collections of computationally intensive problems. As mentioned in Chapter 1, the applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources. HC systems have RMSs (i.e., resource management systems) to govern the execution of tasks that arrive for service. This chapter describes and compares eight heuristics that can be used in such an RMS for dynamically assigning independent tasks to machines.

In a general HC system, schemes are necessary to match tasks to machines, and to schedule the tasks assigned to each machine [6]. Recall from Chapter 1 that the process of matching and scheduling tasks is referred to as mapping. Dynamic methods to do this operate on-line, i.e., as tasks arrive. This is in contrast to static techniques, where the complete set of tasks to be mapped is known *a priori*, the mapping is done off-line, i.e., prior to the execution of any of the tasks, and more time is available to compute the mapping (e.g., [7, 55]).

In the HC environment considered here, the tasks are assumed to be independent, i.e., no communications between the tasks are needed. This scenario is likely to be present, for instance, when many independent users submit their jobs to a collection of shared computational resources. A dynamic scheme is needed because the arrival times of the tasks may be random, and some machines in the suite may go off-line and new machines may come on-line. The dynamic mapping heuristics investigated in this study are non-preemptive, and assume that the tasks have no deadlines or priorities associated with them.

The mapping heuristics can be grouped into two categories: immediate mode and batch mode heuristics. In the immediate mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. As deȚned in Chapter 1, the independent set of tasks that is considered for mapping at the mapping events is called a meta-task. A meta-task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and the ones that were mapped in earlier mapping events but did not begin execution. While immediate mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

The trade-offs among and between immediate and batch mode heuristics are studied experimentally. Mapping independent tasks onto an HC suite is a well-known NP-complete problem if throughput is the optimization criterion [25]. For the heuristics discussed in this chapter, maximization of throughput is the primary objective, because this performance measure is the most common one in production oriented environments.

Three new heuristics, one for batch and two for immediate, are introduced as part of this research. Simulation studies are performed to compare these heuristics with a number of existing ones. In total, Țve immediate mode heuristics and three batch heuristics are examined. The immediate mode heuristics consider, to varying degrees and in different ways, task afȚnity for different machines and machine ready times. The batch mode heuristics consider these factors, as well as aging of tasks waiting to execute. The heuristics developed here, or their derivatives, may be included in the Scheduling Advisor component of the MSHN prototype.

Section 6.2 describes some related work. In Section 6.3, background terms and optimization criteria are deȚned. Section 6.4 discusses the mapping approaches studied here. Section 6.5 gives the simulation procedure, and presents the simulation results.

6.2 Related Work

Related work in literature was examined to select a set of heuristics appropriate for the HC environment considered here, and then perform comparative studies. This section is a sampling of related literature, and is not meant to be exhaustive.

In the literature, mapping tasks onto machines is often referred to as scheduling. Several researchers have worked on the dynamic mapping problem from areas including job shop scheduling and distributed computer systems (e.g., [26, 32, 44, 51]).

The heuristics presented in [25] are concerned with mapping independent tasks onto heterogeneous machines such that the completion time of the last finishing task is minimized. The problem is recognized as NP-complete, and worst case performance bounds are obtained for the heuristics. Some of these heuristics are designed for a general HC environment, while the rest target either a heterogeneous two machine system or a general homogeneous system. Of the heuristics designed for a general HC environment, the A-schedule, B-schedule, and C-schedule heuristics are simply variations of the minimum completion time heuristic used here. The Min-min heuristic that is used here as a benchmark for batch mode mapping is based on the D-schedule, and is also one of the heuristics implemented in SmartNet [19].

The scheme in [26] is representative of techniques for mapping communicating subtasks to an HC suite, considering data dependency graphs and communication times between machines. Thus, an environment very different than the set of independent tasks considered here is used. Hence, the heuristics developed for that different environment are not appropriate for the HC environment considered here.

Two dynamic mapping approaches, one using a distributed policy and the other using a centralized policy, are developed in [32]. Both of these approaches are very similar to the minimum completion time heuristic (used as a benchmark in the studies here) except that they incorporate communication times in calculating the minimum completion time for a task. For the distributed approach, the mapper at a given node considers the local machine and the k highest communication bandwidth neighbors to map the tasks in the local queue. Therefore, the mapper based on the distributed strategy assigns a task to the best machine among the $k + 1$ machines. The simulation results provided in [32] show that the heuristic with the centralized policy always performs better than the distributed heuristic. Hence, the minimum completion time heuristic used here represents the better of the two heuristics presented in [32].

A survey of dynamic scheduling heuristics for job-shop environments is provided in [51]. It classiȚes the dynamic scheduling algorithms into three approaches: knowledge-based approach, conventional approach, and distributed problem solving approach. The heuristics with a knowledge-based approach take a long time to execute, and hence are not suitable for the particular dynamic environment considered here. The classes of heuristics grouped under the conventional and distributed problem solving approaches are similar to the minimum completion time heuristic considered here. However, the problem domains considered in [51] involve precedence constraints among the tasks, priorities, or deadlines, and thus differ from the domain here.

In distributed computer systems, load balancing schemes have been a popular strategy for mapping tasks onto machines (e.g., [39, 44]). In [39], the performance characteristics of simple load balancing heuristics for HC distributed systems are studied. The heuristics presented in [39] do not consider task execution times when making their decisions. In [44], a survey of dynamic scheduling heuristics for distributed computing systems is provided. All heuristics, except one, in [44] schedule tasks on different machines using load sharing techniques, without considering task execution times. (The one heuristic in [44] that does not use load sharing, employs deadlines to schedule tasks, and therefore falls out of the problem domain discussed here.) The load balancing heuristic used in this research is representative of the load balancing techniques in [39] and [44].

SmartNet [19] is an RMS for HC systems that employs various heuristics to map tasks to machines considering resource and task heterogeneity. In this chapter, some SmartNet heuristics appropriate for the HC environment considered here are included in the comparative study (minimum completion time, Min-min, and Max-min).

6.3 Performance Metric

The expected execution time e_{ij} of task t_i on machine m_j is deŢned as the amount of time taken by m_j to execute t_i given m_j has no load when t_i is assigned. The time e_{ij} includes the time to move the t_i code and data from each of their corresponding single Ţxed sources to machine m_j. The expected completion time c_{ij} of task t_i on machine m_j is deŢned as the wall-clock time at which m_j completes t_i (after having Ţnished any previously assigned tasks). Let m be the total number of machines in the HC suite. Let K be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be a_i, and let the time t_i begins execution be b_i. From the above deŢnitions, $c_{ij} = b_i + e_{ij}$. Let c_i be the completion time for task t_i, and is equal to c_{ij} where machine m_j is assigned to execute task t_i. The makespan [41] for the complete schedule is then deŢned as $max_{t_i \in K}(c_i)$. Makespan is a measure of the throughput of the HC system, and does not measure the quality of service imparted to an individual task.

Recall from Section 6.1, that in immediate mode, the mapper assigns a task to a machine as soon as the task arrives at the mapper, and in batch mode a set of independent tasks that need to be mapped at a mapping event is called a meta-task. (In some systems, the term meta-task is deŢned in a way that allows inter-task dependencies.) In batch mode, for the i-th mapping event, the meta-task M_i is mapped at time τ_i, where $i \geq 0$. The initial meta-task, M_0, consists of all the tasks that arrived prior to time τ_0, i.e., $M_0 = \{t_j \mid a_j < \tau_0\}$. The meta-task, M_k, for $k > 0$, consists of tasks that arrived after the last mapping event and the tasks that had been mapped, but did not start executing, i.e., $M_k = \{t_j \mid \tau_{k-1} \leq a_j < \tau_k\} \cup \{t_j \mid a_j < \tau_{k-1}, b_j > \tau_k\}$. Let $\bar{c}_j$ be the completion time of task t_j if it is the only task that is executing on the system. The sharing penalty (ρ_j) for the task t_j is deŢned as $(c_j - \bar{c}_j)$. The average sharing penalty for the tasks in the set K is given by $[\sum_{t_j \in K} \rho_j] / \mid K \mid$. The average sharing penalty for a set of tasks mapped by a given heuristic is an indication of the heuristic's ability to minimize the effects of contention among different tasks in the set. It indicates quality of service provided to an individual task, as gauged by the wait incurred by the task before it begins and the time to perform the actual computation.

6.4 Mapping Heuristics

6.4.1 Overview

In the immediate mode heuristics, each task is considered only once for matching and scheduling, i.e., the mapping is not changed once it is computed. When the arrival rate is low enough, machines may be ready to execute a task as soon as it arrives at the mapper. Therefore, it may be beneŢcial to use the mapper in the immediate mode so that a task need not wait until the next mapping event to begin its execution.

In batch mode, the mapper considers a meta-task for matching and scheduling at each mapping event. This enables the mapping heuristics to possibly make better decisions than immediate mode heuristics. This is because the batch mode heuristics have the resource requirement information for a whole meta-task, and know about the actual execution times of a larger number of tasks (as more tasks might complete while waiting for the mapping event). When the task arrival rate is high, there will be a sufŢcient number of tasks to keep the machines busy in between the mapping events, and while a mapping is being computed. (It is, however, assumed in this study that the running time of each mapping heuristic is negligibly small as compared to the average task execution time.)

Both immediate and batch mode heuristics assume that estimates of expected task execution times on each machine in the HC suite are known (see Section 4.2). These estimates can be supplied before a task is submitted for execution, or at the time it is submitted.

The <u>ready</u> <u>time</u> of a machine is the earliest wall-clock time that machine is going to be ready after completing the execution of tasks that are currently assigned to it. Because the heuristics presented here are dynamic, the expected machine ready times are based on a combination of actual task execution times (for tasks that have completed execution on that machine) and estimated expected task execution times (for tasks assigned to that machine and waiting to execute). It is assumed that each time a task t_i completes on a machine m_j a report is sent to the mapper, and the ready time for m_j is updated if necessary. The experiments presented in Section 6.5 model this situation using simulated actual values for the execution times of tasks that have already Ţnished their execution.

All heuristics examined here operate in a centralized fashion and map tasks onto a dedicated suite of machines; i.e., the mapper controls the execution of all jobs on all machines in the suite. It is assumed that each mapping heuristic is being run on a separate machine. (While all heuristics studied here are functioning dynamically, the use of some of these heuristics in a static environment is discussed in [7], and is also summarized in Chapter 7 as related work.)

6.4.2 Immediate mode mapping heuristics

Five immediate mode heuristics are described here. These are (i) minimum completion time, (ii) minimum execution time, (iii) switching algorithm, (iv) k-percent best, and (v) opportunistic load balancing. Of these Ţve heuristics, switching algorithm and k-percent best have been proposed as part of the research presented in this thesis.

The minimum completion time (MCT) heuristic assigns each task to the machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The MCT heuristic is a variant of the fast-greedy heuristic from SmartNet [19]. The MCT heuristic is used as a benchmark for the immediate mode, i.e., the performance of other heuristics is compared with that of the MCT heuristic. As a task arrives, all the machines in the HC suite are examined to determine the machine that gives the earliest completion time for the task. Therefore, it takes $O(m)$ time to map a given task.

The minimum execution time (MET) heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time (this heuristic is also known as limited best assignment (LBA) [1] and user directed assignment (UDA) [19]). This heuristic, in contrast to MCT, does not consider machine ready times, and can cause a severe imbalance in load across the machines. The advantages of this method are that it gives each task to the machine that performs it in the least amount of execution time, and the heuristic is very simple. The heuristic needs $O(m)$ time to Ţnd the machine that has the minimum execution time for a task.

The switching algorithm (SA) is motivated by the following observations. The MET heuristic can potentially create load imbalance across machines by assigning many more tasks to some machines than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. If the tasks are arriving in a random mix, it is possible to use the MET at the expense of load balance until a given threshold, and then use the MCT to smooth the load across the machines. The SA heuristic uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. The purpose is to have a heuristic with the desirable properties of both the MCT and the MET.

Let the maximum (latest) ready time over all machines in the suite be $\underline{r_{max}}$, and the minimum (earliest) ready time be $\underline{r_{min}}$. Then, the load balance index across the machines is given by $\underline{\pi} = r_{min}/r_{max}$. The parameter π can have any value in the interval $[0, 1]$. If π is 1.0, then the load is evenly balanced across the machines. If π is 0, then at least one machine has not yet been assigned a task. Two threshold values, $\underline{\pi_l}$ (low) and $\underline{\pi_h}$ (high), for the ratio π are chosen in $[0, 1]$ such that $\pi_l < \pi_h$. Initially, the value of π is set to 0.0. The SA heuristic begins mapping tasks using the MCT heuristic until the value of the load balance index increases to at least π_h. After that point in time, the SA heuristic begins using the MET heuristic to perform task mapping. This typically causes the load balance index to decrease. When it decreases to π_l or less, the SA heuristic switches back to using the MCT heuristic for mapping the tasks, and the cycle continues.

As an example of the functioning of the SA with lower and upper limits of 0.6 and 0.9, respectively, for $\mid K \mid$=1000 and one particular rate of arrival of tasks, the SA switched between the MET and the MCT two times (i.e., from the MCT to the MET to the MCT), assigning 715 tasks using the MCT. For $| K |$=2000 and the same task arrival rate, the SA switched Ţve times, using the MCT to assign 1080 tasks. The percentage of tasks assigned using MCT gets progressively smaller for larger $| K |$. This is because the larger the $| K |$, the larger the number of tasks waiting to execute on a given machine, and therefore, the larger the ready time of a given machine. This in turn means that an arriving task's execution time will impact the machine ready time less, thereby rendering the load balance index less sensitive to a load-imbalancing assignment.

At each task's arrival, the SA heuristic determines the load balance index. In the worst case, this takes $O(m)$ time. In the next step, the time taken to assign a task to a machine is of order $O(m)$, whether SA uses the MET to perform the mapping or the MCT. Overall, the SA heuristic takes $O(m)$ time irrespective of which heuristic is actually used for mapping the task.

The k-percent best (KPB) heuristic considers only a subset of machines while mapping a task. The subset is formed by picking the $m \times (k/100)$ best machines based on the execution times for the task, where $100/m \leq k \leq 100$. The task is assigned to a machine that provides the earliest completion time in the subset. If $k = 100$, then the KPB heuristic is reduced to the MCT heuristic. If $k = 100/m$, then the KPB heuristic is reduced to the MET heuristic. A "good" value of k maps a task to a machine only within a subset formed from computationally superior machines. The purpose is not as much as matching of the current task to a computationally well-matched machine as it is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks. This "foresight" about task heterogeneity is missing in the MCT, which might assign a task to a poorly matched machine for an local marginal improvement in completion time, possibly depriving some subsequently arriving better matched tasks of that machine, and eventually leading to a larger makespan as compared to the KPB. It should be noted that while both the KPB and SA combine elements of the MCT and the MET in their operation, it is only in the KPB that *each* task assignment attempts to optimize objectives of the MCT and the

Table 6.1 Initial ready times of the machines (arbitrary units).

m_0	m_1	m_2
75	110	200

MET simultaneously. However, in cases where a ﬁxed subset of machines is not among the $k\%$ best for any of the tasks, the KPB will cause more machine idle time compared to the MCT, and can result in much poorer performance. Thus the relative performance of the KPB and the MCT may depend on the HC suite of machines, and characteristics of the tasks being executed.

For each task, $O(m \log m)$ time is spent in ranking the machines for determining the subset of machines to examine. Once the subset of machines is determined, it takes $O(\frac{m \times k}{100})$ time, i.e., $O(m)$ time to determine the machine assignment. Overall the KPB heuristic takes $O(m \log m)$ time.

The opportunistic load balancing (OLB) heuristic assigns a task to the machine that becomes ready next, without considering the execution time of the task onto that machine. If multiple machines become ready at the same time, then one machine is arbitrarily chosen. The complexity of the OLB heuristic is dependent on the implementation. In the implementation considered here, the mapper may need to examine all m machines to ﬁnd the machine that becomes ready next. Therefore, it takes $O(m)$ to ﬁnd the assignment. Other implementations may require idle machines to assign tasks to themselves by accessing a shared global queue of tasks [54].

As an example of the working of these heuristics, consider a simple system of three machines, m_0, m_1, and m_2, currently loaded so that expected ready times are as given in Table 6.1. Consider the performance of the heuristics for a very simple case of three tasks t_0, t_1, and t_2 arriving in that order. Table 6.2 shows the expected execution times of tasks on the machines in the system. All time values in the discussion below are the expected values.

Table 6.2 Expected execution times (arbitrary units).

	m_0	m_1	m_2
t_0	50	20	15
t_1	20	60	15
t_2	20	50	15

The MET Ţnds that all tasks have their minimum completion time on m_2, and even though m_2 is already heavily loaded, it assigns all three tasks to m_2. The time when t_0, t_1, and t_2 will all have completed is 245 units.

The OLB assigns t_0 to m_0 because m_0 is expected to be idle soonest. Similarly, it assigns t_1 and t_2 to m_1 and m_0, respectively. The time when t_0, t_1, and t_2 will all have completed is 170 units.

The MCT determines that the minimum completion time for t_0 will be achieved on m_0, and makes this assignment, even though the execution time of t_0 on m_0 is more than twice of that on m_1 (where the completion time would have been only slightly larger). Then MCT goes on to assign t_1 to m_0, and t_2 to m_1 so that the time when t_0, t_1, and t_2 will all have completed is 160 units.

The SA Ţrst determines the current value of the load balance index, π, which is $75/200$ or 0.38. Assume that π_l is 0.40 and that π_h is 0.70. Because $\pi < \pi_l$, the SA chooses the MCT to make the Ţrst assignment. After the Ţrst assignment, $\pi = 110/200 = 0.55 < \pi_h$. So the SA continues to use the MCT for the second assignment as well. It is only after the third assignment that $\pi = 145/200 = 0.725 > \pi_h$, so the SA will then use the MET for the fourth arriving task. The time when t_0, t_1, and t_2 will all have completed here is the same as that for the MCT.

Let the value of k in the KPB be 67% so that the KPB will choose from the two fastest executing machines to assign a given task. For t_0, these machines are m_1 and m_2. Within these two machines, the minimum completion time is achieved on m_1 so t_0 is assigned to

m_1. This is the major difference from the working of the MCT; m_0 is not assigned t_0 even though t_0 would have its minimum completion time (over all machines) there. This step saves m_0 for any yet-to-arrive tasks that may run slowly on other machines. One such task is t_2; in the MCT it is assigned to m_1, but in the KPB it is assigned to m_0. The time when t_0, t_1, and t_2 will all have completed using the KPB is 130 units. This is the smallest among all Ţve heuristics.

6.4.3 Batch mode mapping heuristics

Three batch mode heuristics are described here: (i) the Min-min heuristic, (ii) the Max-min heuristic, and (iii) the Sufferage heuristic. The Sufferage heuristic has been proposed as part of the research presented in this thesis. In the batch mode heuristics, meta-tasks are mapped after predeŢned intervals. These intervals are deŢned in this study using one of the two strategies proposed below.

The regular time interval strategy maps the meta-tasks at regular intervals of time (e.g., every ten seconds). The only occasion when such a mapping will be redundant is when: (1) no new tasks have arrived since the last mapping, and (2) no tasks have Ţnished executing since the last mapping (thus, machine ready times are unchanged). These conditions can be checked for, and so redundant mapping events can be avoided.

The Ţxed count strategy maps a meta-task M_i as soon as one of the following two mutually exclusive conditions are met: (a) an arriving task makes $\mid M_i \mid$ larger than or equal to a predetermined arbitrary number $\underline{\kappa}$, or (b) all tasks in the set $\mid K \mid$ have arrived, and a task completes while the number of tasks that yet have to begin is greater than or equal to κ. In this strategy, the time between the mapping events will depend on the arrival rate and the completion rate. The possibility of machines being idle while waiting for the next mapping event will depend on the arrival rate, completion rate, m, and κ. (For the arrival rates in the experiments here, this strategy operates reasonably; in an actual system, it may be necessary for tasks to have a maximum waiting time to be mapped.)

The batch mode heuristics considered in this study are discussed in the paragraphs below. The complexity analysis performed for these heuristics considers a single mapping event, and the meta-task size is assumed to be equal to the average of meta-task sizes at all actually performed mapping events. Let the average meta-task size be S.

The Min-min heuristic shown in Figure 6.1 is from [25], and is one of the heuristics implemented in SmartNet [19]. In Figure 6.1, let r_j denote the expected time machine m_j will become ready to execute a task after Ţnishing the execution of all tasks assigned to it at that point in time. First the c_{ij} entries are computed using the e_{ij} and r_j values. For each task t_i, the machine that gives the earliest expected completion time is determined by scanning the i-th row of the c matrix (composed of the c_{ij} values). The task t_k that has the minimum earliest expected completion time is determined and then assigned to the corresponding machine. The matrix c and vector r are updated and the above process is repeated with tasks that have not yet been assigned a machine.

Min-min begins by scheduling the tasks that change the expected machine ready time status by the least amount. If tasks t_i and t_k are contending for a particular machine m_j, then Min-min assigns m_j to the task (say t_i) that will change the ready time of m_j less. This increases the probability that t_k will still have its earliest completion time on m_j, and shall be assigned to it. Because at $t = 0$, the machine that Ţnishes a task earliest is also the one that executes it fastest, and from thereon the Min-min heuristic changes machine ready time status by the least amount for every assignment, the percentage of tasks assigned their best choice (PTBC) (on basis of expected execution time) is likely to be higher in Min-min than with the other batch mode heuristics described in this section (this has been veriŢed by examining the simulation study data given later in this thesis). The expectation is that a smaller makespan can be obtained if a larger number of tasks is assigned to the machines that not only complete them earliest, but also execute them fastest.

```
(1)  for all tasks $t_i$ in meta-task $M_v$ (in an arbitrary order)
(2)       for all machines $m_j$ (in a fixed arbitrary order)
(3)            $c_{ij} = e_{ij} + r_j$
(4)  do until all tasks in $M_v$ are mapped
(5)       for each task in $M_v$ find the earliest completion
               time and the machine that obtains it
(6)       find the task $t_k$ with the minimum earliest
               completion time
(7)       assign task $t_k$ to the machine $m_l$ that gives the
(8)            earliest completion time
(9)       delete task $t_k$ from $M_v$
(10)      update $r_l$
(11)      update $c_{il}$ for all $i$
(12) enddo
```

Fig. 6.1. The Min-min heuristic.

The initialization of the c matrix in Line (1) to Line (3) of Figure 6.1 takes $O(Sm)$ time. The **do** loop of the Min-min heuristic is repeated S times and each iteration takes $O(Sm)$ time. Therefore, the heuristic takes $O(S^2m)$ time.

The Max-min heuristic is similar to the Min-min heuristic, and is one of the heuristics implemented in SmartNet [19]. It differs from the Min-min heuristic (given in Figure 6.1) in that once the machine that provides the earliest completion time is found for every task, the task t_k that has the maximum earliest completion time is determined and then assigned to the corresponding machine. That is, in Line (6) of Figure 6.1, “minimum” would be changed to “maximum.” The Max-min heuristic has the same complexity as the Min-min heuristic.

The Max-min is likely to do better than the Min-min heuristic in cases where there are many more shorter tasks than longer tasks. For example, if there is only one long task, Max-min will execute many short tasks concurrently with the long task. The resulting makespan may just be determined by the execution time of the long task in this case. Min-min, however, Ţrst Ţnishes the shorter tasks (which may be more or less evenly distributed over the machines) and then executes the long task, increasing the makespan compared to the Max-min.

The Sufferage heuristic (shown in Figure 6.2) is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it. Let the sufferage value of a task t_i be the difference between its second earliest completion time (on some machine m_y) and its earliest completion time (on some machine m_x). That is, using m_x will result in the best completion time for t_i, and using m_y the second best.

The initialization phase in Lines (1) to (3), in Figure 6.2, is similar to the ones in the Min-min and Max-min heuristics. Initially all machines are marked "unassigned." In each iteration of the **for** loop in Lines (6) to (14), pick arbitrarily a task t_k from the meta-task. Find the machine m_j that gives the earliest completion time for task t_k, and tentatively assign m_j to t_k if m_j is unassigned. Mark m_j as assigned, and remove t_k from meta-task. If, however, machine m_j has been previously assigned to a task t_i, choose from t_i and t_k the task that has the higher sufferage value, assign m_j to the chosen task, and remove the chosen task from the meta-task. The unchosen task will not be considered again for this execution of the **for** statement, but shall be considered for the next iteration of the **do** loop beginning on Line (4). When all the iterations of the **for** loop are completed (i.e., when one execution of the **for** statement is completed), update the machine ready time of each machine that is assigned a new task. Perform the next iteration of the **do** loop beginning on Line (4) until all tasks have been mapped.

Table 6.3 shows a scenario in which the Sufferage will outperform the Min-min. Table 6.3 shows the expected execution time values for four tasks on four machines (all initially idle). In this case, the Min-min heuristic gives a makespan of 93 and the Sufferage heuristic gives a makespan of 78. Figure 6.3 gives a pictorial representation of the assignments made for the case in Table 6.3.

From the pseudo-code given in Figure 6.2, it can be observed that the ﬁrst execution of the **for** statement on Line (6) takes $O(Sm)$ time. The number of task assignments made in one execution of this **for** statement depends on the total number of machines in the HC suite, the number of machines that are being contended for among different tasks, and the number of tasks in the meta-task being mapped. In the worst case, only one task assignment will be made in each execution of the **for** statement. Then meta-task size will decrease by one at each **for** statement execution. The outer **do** loop will be iterated S times to map the entire meta-task. Therefore, in the worst case, the time $T(S)$ taken to map a meta-task of size S will be

$$T(S) = Sm + (S-1)m + (S-2)m + \cdots + m$$

$$T(S) = O(S^2 m)$$

In the best case, there are as many machines as there are tasks in the meta-task, and there is no contention among the tasks. Then all the tasks are assigned in the ﬁrst execution of the **for** statement so that $T(S) = O(Sm)$. Let $\underline{\omega}$ be a number quantifying the extent of contention among the tasks for the different machines. The complexity of the Sufferage heuristic can then be given as $O(\omega Sm)$, where $1 \leq \omega \leq S$. It can be seen that ω is equal to S in the worst case, and is 1 in the best case; these values of ω are numerically equal to the number of iterations of the **do** loop on Line (4), for the worst and the best case, respectively.

Table 6.3 An example expected execution time matrix that illustrates the situation where the Sufferage heuristic outperforms the Min-min heuristic.

	m_0	m_1	m_2	m_3
t_0	40	48	134	50
t_1	50	82	88	89
t_2	55	68	94	93
t_3	52	60	78	108

The batch mode heuristics can cause some tasks to be starved of machines. Let $\underline{H_i}$ be a subset of meta-task M_i consisting of tasks that were mapped (as part of M_i) at the mapping event i at time τ_i but did not begin execution by the next mapping event at τ_{i+1}. H_i is the subset of M_i that is included in M_{i+1}. Due to the expected heterogeneous nature of the tasks, the meta-task M_{i+1} may be mapped so that some or all of the tasks arriving between τ_i and τ_{i+1} may begin executing before the tasks in set H_i do. It is possible that some or all of the tasks in H_i may be included in H_{i+1}. This probability increases as the number of new tasks arriving between τ_i and τ_{i+1} increases. In general, some tasks may be remapped at each successive mapping event without actually beginning execution (i.e., the task is starving for a machine). This impacts the response time the user sees.

To reduce starvation, aging schemes are implemented. The age of a task is set to zero when it is mapped for the Ţrst time and incremented by one each time the task is remapped. Let σ be a constant that can be adjusted empirically to change the extent to which aging affects the operation of the heuristic. An aging factor, $\zeta = (1 + \mathrm{age}/\sigma)$, is then computed for each task. For the experiments in this study, σ is arbitrarily set to 10 (e.g., in this case, the aging factor for a task increases by one after every ten remappings of the task). The aging factor is used to enhance the probability of an "older" task beginning before the tasks that would otherwise begin Ţrst. In the Min-min heuristic, for each task, the completion time obtained in Line (5) of Figure 6.1 is multiplied by the corresponding value for $\frac{1}{\zeta}$. As the age of a task increases, its age-compensated expected completion time (i.e., the value used to determine the mapping) gets increasingly smaller than its original expected completion time. This increases its probability of being selected in Line (6) in Figure 6.1.

For the Max-min heuristic, the completion time of a task is multiplied by ζ. In the Sufferage heuristic, the sufferage value computed in Line (8) in Figure 6.2 is multiplied by ζ.

6.5 Experimental Results and Discussion

6.5.1 Simulation Procedure

The mappings are simulated using a discrete event simulator ([8, 27, 42]). The task arrivals are modeled by a Poisson random process. Recall that the simulator contains the ETC matrix that contains the expected execution times of a task on all machines, for all the tasks that can arrive for service. The ETC matrix entries used in the simulation studies represent the e_{ij} values (in seconds) that the heuristic would use in its operation. The actual execution time of a task can be different than the value given by the ETC matrix. This variation is modeled by generating a simulated actual execution time for each task by sampling a Gaussian probability density function with variance equal to three times the expected execution time of the task and mean equal to the expected execution time of the task (e.g., [2, 40]). If the sampling results in a negative value, the value is discarded and the same probability density function is sampled again (i.e., a truncated Gaussian distribution is sampled). This process is repeated until a positive value is returned by the sampling process.

In the experiments described here, the ETC matrix is generated with the range-based method. (This method has been explained in Chapter 4.) The values of R_t (see Chapter 4) for low and high task heterogeneities are 1000 and 3000, respectively. The values of R_m (see Chapter 4) for low and high machine heterogeneities are 10 and 100, respectively. These heterogeneity ranges are based on one type of expected environment for MSHN.

The experimental evaluation of the heuristics is performed in three parts. In the first part, the immediate mode heuristics are compared using makespan and average sharing penalty. The second part involves a comparison of the batch mode heuristics. The third part is the comparison of the batch mode and the immediate mode heuristics. Unless stated otherwise, the following are valid for the experiments described here. The number of machines is held constant at 20, and the experiments are performed for $\mid K \mid \; = \{1000,\ 2000\}$. All heuristics are evaluated in a HiHi heterogeneity environment, both for the inconsistent and the semi-consistent cases, because these correspond to some of the currently expected MSHN environments.

For each value of $\mid K \mid$, tasks are mapped under two different Poisson arrival rates, $\underline{\lambda_h}$ and $\underline{\lambda_l}$, such that $\lambda_h > \lambda_l$. The value of λ_h is chosen empirically to be high enough to allow at most 50% tasks to have completed when the last task in the set arrives. That is, for λ_h, when at least 50% of the tasks execute no new tasks are arriving. This may correspond to a situation when tasks are submitted during the day but not at night.

In contrast, λ_l is chosen to be low enough to allow at least 90% of the tasks to have completed when the last task in the set arrives. That is, for λ_l, when at most 10% of the tasks execute no new tasks are arriving. This may correspond more closely than λ_h to a situation where tasks arrive continuously. The difference between λ_h and λ_l can also be considered to represent a difference in burstiness.

Some experiments were also performed at a third arrival rate $\underline{\lambda_t}$, where λ_t was high enough to ensure that only 20% of the tasks have completed when the last task in the set arrived. The MCT heuristic was used as a basis for these percentages. Unless otherwise stated, the task arrival rate is set to λ_h.

Example comparisons are discussed in Subsections 6.5.2 to 6.5.4. Each data point in the comparison charts is an average over 50 trials, where for each trial the simulated actual task execution times are chosen independently. The values of performance metrics for each trial for each heuristic have been normalized with respect to the benchmark heuristic, which is the MCT for immediate mode heuristics, and the Min-min for the batch mode heuristics. The Min-min serves as a benchmark also for the experiments where batch mode heuristics are compared with immediate mode heuristics. Each bar (except the one for the benchmark heuristic) in the comparison charts gives a 95% conȚdence interval (shown as an "I" on the top of the bars) for the mean of the normalized value. Occasionally upper bound, lower bound, or the entire conȚdence interval is not distinguishable from the mean value, for the scale used in the graphs here. More general conclusions about the heuristics' performance are in Section 8.

6.5.2 Comparisons of the immediate mode heuristics

Unless otherwise stated, the immediate mode heuristics are investigated under the following conditions. In the KPB heuristic, k is equal to 20%. This particular value of k was found to give the lowest makespan for the KPB heuristic under the conditions of the experiments. For the SA, the lower threshold and the upper threshold for the load balance index are 0.6 and 0.9, respectively. Once again these values were found to give optimum values of makespan for the SA.

In Figure 6.4, immediate mode heuristics are compared based on normalized makespan for inconsistent HiHi heterogeneity. From Figure 6.4, it can be noted that the KPB heuristic completes the execution of the last Ţnishing task earlier than the other heuristics (however, it is only slightly better than the MCT). For $k = 20\%$ and m = 20, the KPB heuristic forces a task to choose a machine from a subset of four machines. These four machines have the lowest execution times for the given task. The chosen machine would give the smallest completion time as compared to other machines in the set.

Figure 6.5 compares the immediate mode heuristics using normalized average sharing penalty. Once again, the KPB heuristic performs best. However, the margin of improvement is smaller than that for the makespan. It is evident that the KPB provides maximum throughput (system oriented performance metric) and minimum average sharing penalty (application oriented performance metric). Figure 6.6 compares the normalized makespans of the different immediate mode heuristics for semi-consistent HiHi heterogeneity. Figure 6.7 compares the normalized average sharing penalties of the different immediate mode heuristics. As shown in Figures 6.4 and 6.6, the relative performance of the different immediate mode heuristics is impacted by the degree of consistency of the ETC matrices. However, the KPB still performs best, closely followed by the MCT.

For the semi-consistent type of heterogeneity, machines within a particular subset perform tasks that lie within a particular subset faster than other machines. From Figure 6.6, it can be observed that for semi-consistent ETC matrices, the MET heuristic performs the worst. For the semi-consistent matrices used in these simulations, the MET heuristic maps half of the tasks to the same machine, considerably increasing the load imbalance. Although the KPB considers only the fastest four machines for each task for the particular value of k used here (which happen to be the same four machines for half of the tasks), the performance does not differ much from the inconsistent HiHi case. Additional experiments have shown that the KPB performance is quite insensitive to values of k as long as k is larger than the minimum value (where the KPB heuristic is reduced to the MET heuristic). For example, when k is doubled from its minimum value of 5%, the makespan decreases by a factor of about Ṭve. However a further doubling of k brings down the makespan by a factor of only about 1.2.

6.5.3 Comparisons of the batch mode heuristics

Figures 6.8 and 6.9 compare the batch mode heuristics based on normalized makespan and normalized average sharing penalty, respectively. In these comparisons, unless otherwise stated, the regular time interval strategy is employed to schedule meta-task mapping events. The time interval is set to 10 seconds. This value was empirically found to optimize makespan for the Min-min over other values. From Figure 6.8, it can be noted that the Sufferage heuristic outperforms the Min-min and the Max-min heuristics based on makespan (although it is only slightly better than the Min-min). However, for average sharing penalty, the Min-min heuristic outperforms the other heuristics (Figure 6.9). The Sufferage heuristic considers the “loss” in completion time of a task if it is not assigned to its first choice in making the mapping decisions. By assigning their first choice machines to the tasks that have the highest sufferage values among all contending tasks, the Sufferage heuristic reduces the overall completion time.

Furthermore, it can be noted that the makespan given by the Max-min is much larger than the makespans obtained by the other two heuristics. Using reasoning similar to that given in Subsection 6.4.3 for explaining better expected performance for the Min-min, it can be seen that the Max-min assignments change a given machine’s ready time status by a larger amount than the Min-min assignments do. If tasks t_i and t_k are contending for a particular machine m_j, then the Max-min assigns m_j to the task (say t_i) that will increase the ready time of m_j more. This decreases the probability that t_k will still have its earliest completion time on m_j and shall be assigned to it. Experimental data (e.g., that given in Figure 6.10) shows that the percentage of tasks assigned their best machine is likely to be lower for the Max-min than for other batch mode heuristics. It might be expected that a larger makespan will result if a larger number of tasks is assigned to the machines that do not have the best execution times for those tasks.

Figure 6.11 compares the makespan of the batch mode heuristics for semi-consistent HiHi heterogeneity. The comparison of the same heuristics for the same parameters is shown in Figure 6.12 with respect to normalized average sharing penalty, and in Figure 6.13 with respect to percentage of tasks assigned their best machine. It can be seen that the results for semi-consistent HiHi are similar to those for inconsistent HiHi.

The impact of aging on batch mode heuristics is shown in Figures 6.14 and 6.15. The Min-min without aging is used here to normalize the performance of the other heuristics.

The Max-min beneŢts most from the aging scheme. Recall that the Min-min performs much better than the Max-min when there is no aging. Aging modiŢes the Max-min's operation so that tasks with smaller completion times can be scheduled prior to those with larger completion times, thus reducing the negative aspects of that technique. This is discussed further in [35].

Figures 6.16, 6.17, 6.18, and 6.19 show the result of repeating the above experiments with a Ţxed count strategy for a batch size of 40. This particular batch size was found to give an optimum value of the makespan for the Min-min heuristic. The Min-min with regular time interval strategy (interval of ten seconds) is used here to normalize the performance of the other heuristics. Figure 6.16 compares the regular time interval and Ţxed count strategies on the basis of normalized makespans given by different heuristics for inconsistent HiHi heterogeneity. In Figure 6.17, the normalized average sharing penalties of the same heuristics for the same parameters are compared. It can be seen that the Ţxed count approach gives similar results for the Min-min and the Sufferage heuristics. The Max-min heuristic, however, beneŢts considerably from the Ţxed count approach; makespan drops to about 60% for $\mid K \mid$=1000, and to about 50% for $\mid K \mid= 2000$ as compared to the makespan given by the regular time interval strategy. A possible explanation lies in a conceptual element of similarity between the Ţxed count approach and the aging scheme. The value of κ =40 used here resulted in batch sizes that were smaller than those using the ten second regular time interval strategy. Thus, small tasks waiting to execute will have fewer tasks to compete with, and, hence, less chance of being delayed by a larger task. Figures 6.18 and 6.19 show the normalized makespan and the normalized average sharing penalty for the semi-consistent case. As compared to the inconsistent case, the regular time interval approach gives slightly better results than the Ţxed count approach for the Sufferage and the Min-min. For the Max-min, however, for both inconsistent and semi-consistent cases, the Ţxed count strategy gives a much larger improvement over the regular time strategy.

It should be noted that all the results given here are for HiHi heterogeneity. Results may differ for other types of heterogeneity. For example, for LoLo heterogeneity, the performance of the Max-min is almost identical to that of the Min-min [35].

```
(1)  for all tasks $t_k$ in meta-task $M_v$ (in an arbitrary order)
(2)       for all machines $m_j$ (in a fixed arbitrary order)
(3)            $c_{kj} = e_{kj} + r_j$
(4)  do until all tasks in $M_v$ are mapped
(5)       mark all machines as “unassigned”
(6)       for each task $t_k$ in $M_v$ (in a fixed arbitrary order)
               /* for a given execution of the for statement,
               each $t_k$ in $M_v$ is considered only once */
(7)            find machine $m_j$ that gives the earliest
                    completion time
(8)            sufferage value $=$ second earliest completion
                    time $-$ earliest completion time
(9)            if machine $m_j$ is unassigned
(10)                assign $t_k$ to machine $m_j$, delete $t_k$
                    from $M_v$, mark $m_j$ assigned
(11)           else
(12)                if sufferage value of task $t_i$ already
                    assigned to $m_j$ is less than the
                    sufferage value of task $t_k$
(13)                     unassign $t_i$, add $t_i$ back to $M_v$,
                         assign $t_k$ to machine $m_j$,
                         delete $t_k$ from $M_v$
(14)      endfor
(15)      update the vector $r$ based on the tasks that
               were assigned to the machines
(16)      update the $c$ matrix
(17) enddo
```

Fig. 6.2. The Sufferage heuristic.

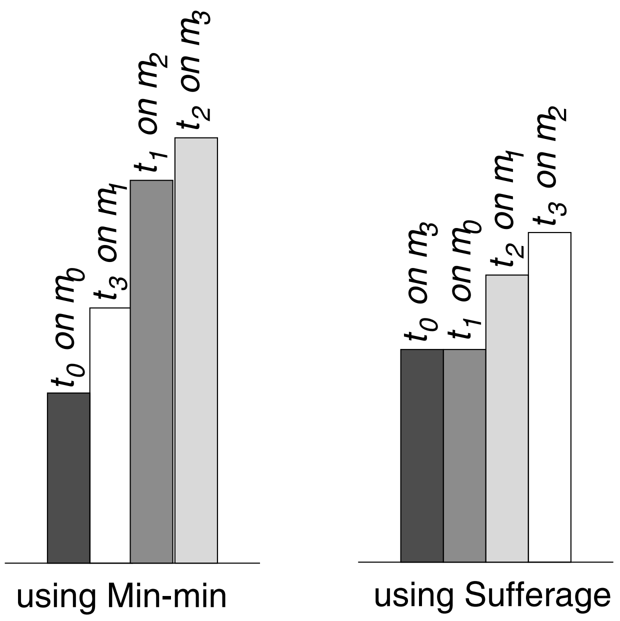

Fig. 6.3. An example scenario (based on Table 6.3) where the Sufferage gives a shorter makespan than the Min-min (bar heights are proportional to task execution times).

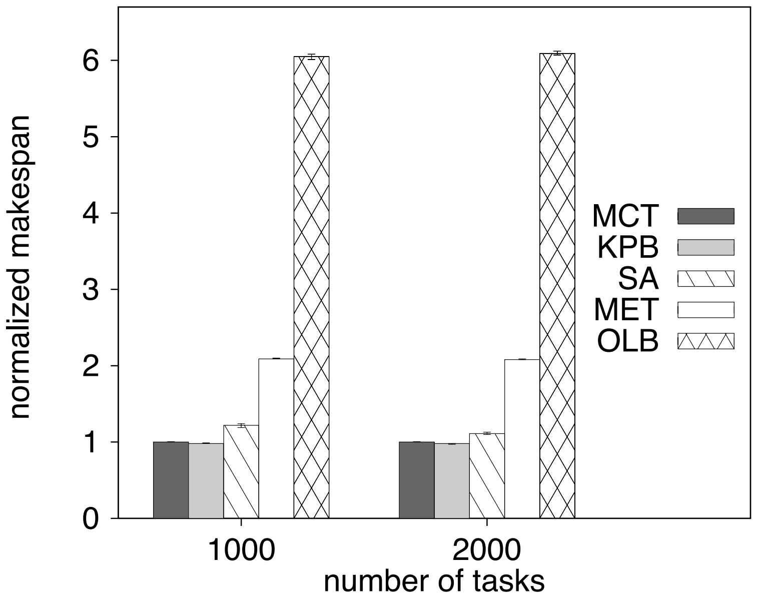

Fig. 6.4. Makespan for the immediate mode heuristics for inconsistent HiHi heterogeneity.

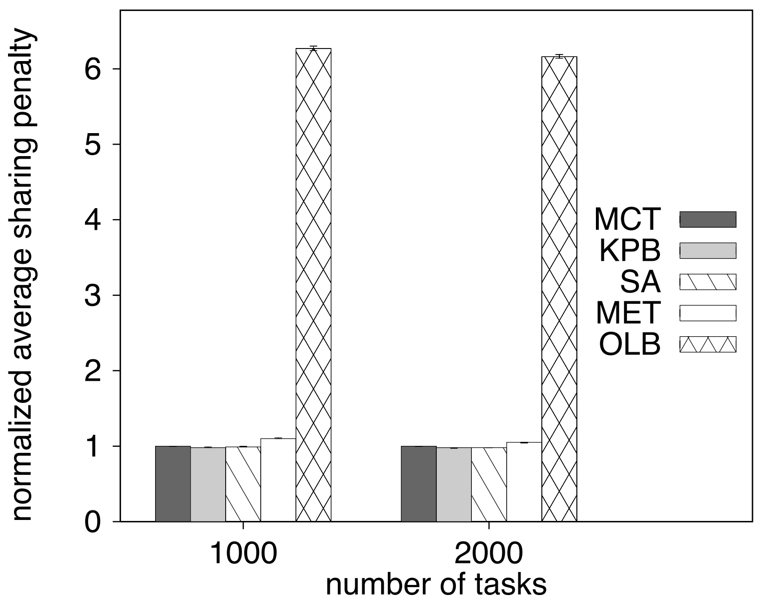

Fig. 6.5. Average sharing penalty of the immediate mode heuristics for inconsistent HiHi heterogeneity.

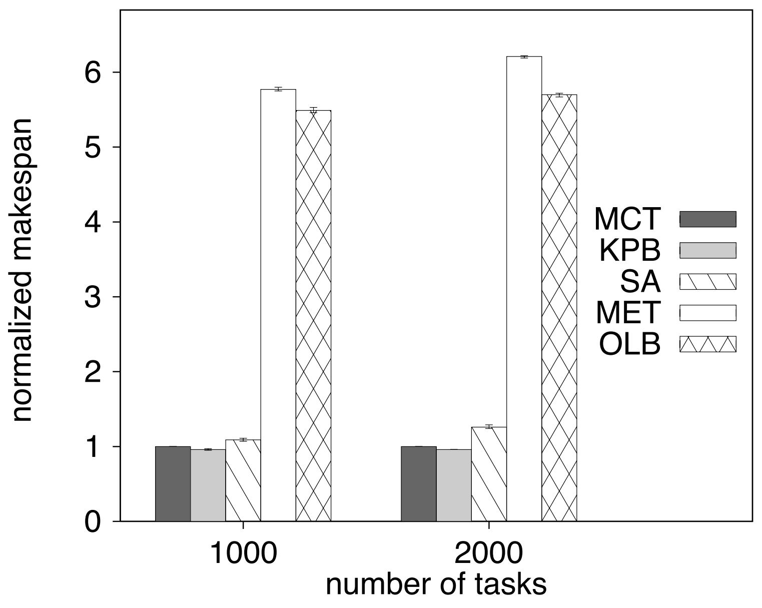

Fig. 6.6. Makespan of the immediate mode heuristics for semi-consistent HiHi heterogeneity.

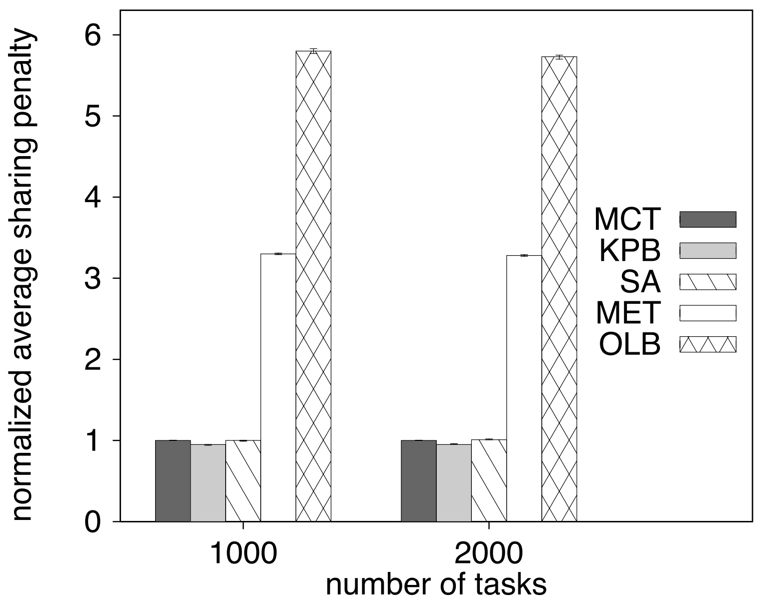

Fig. 6.7. Average sharing penalty of the immediate mode heuristics for semi-consistent HiHi heterogeneity.

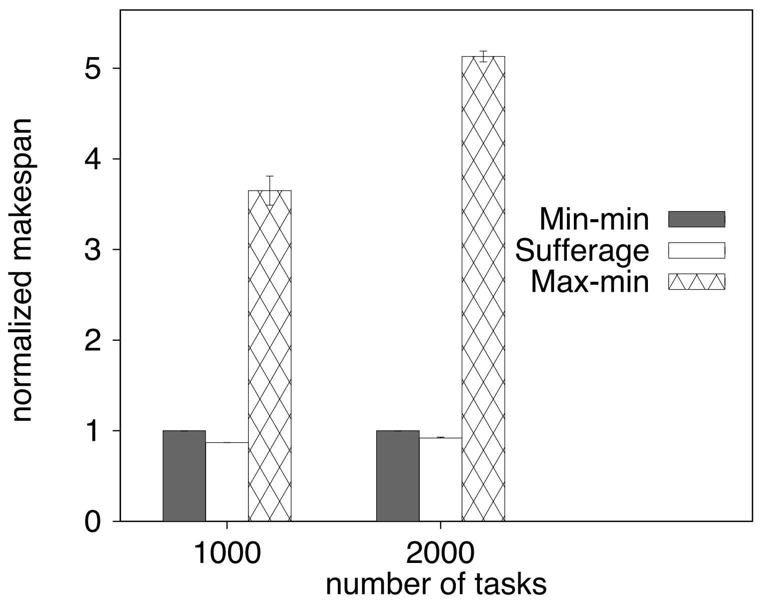

Fig. 6.8. Makespan of the batch mode heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity.

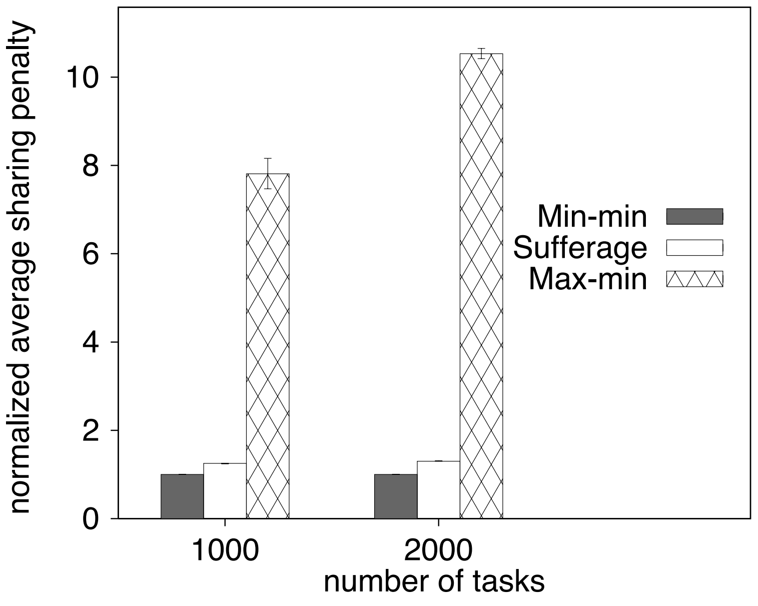

Fig. 6.9. Average sharing penalty of the batch mode heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity.

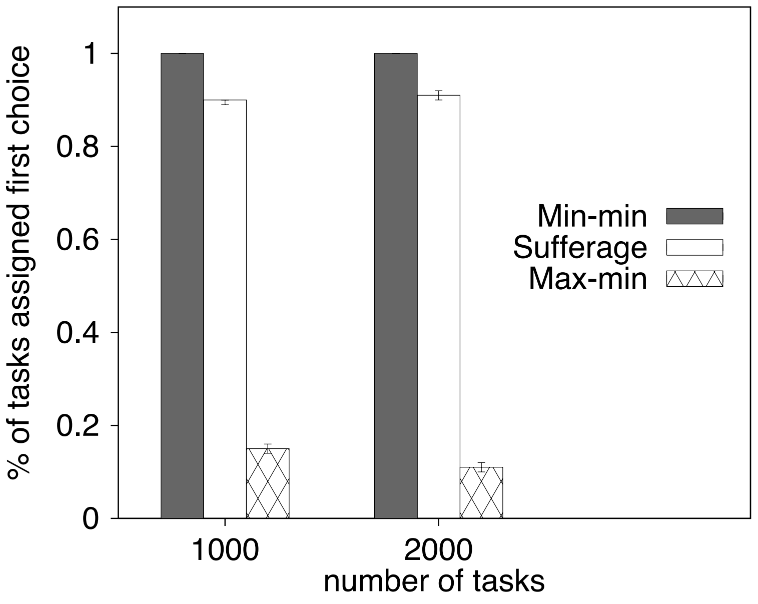

Fig. 6.10. PTBC for the batch mode heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity.

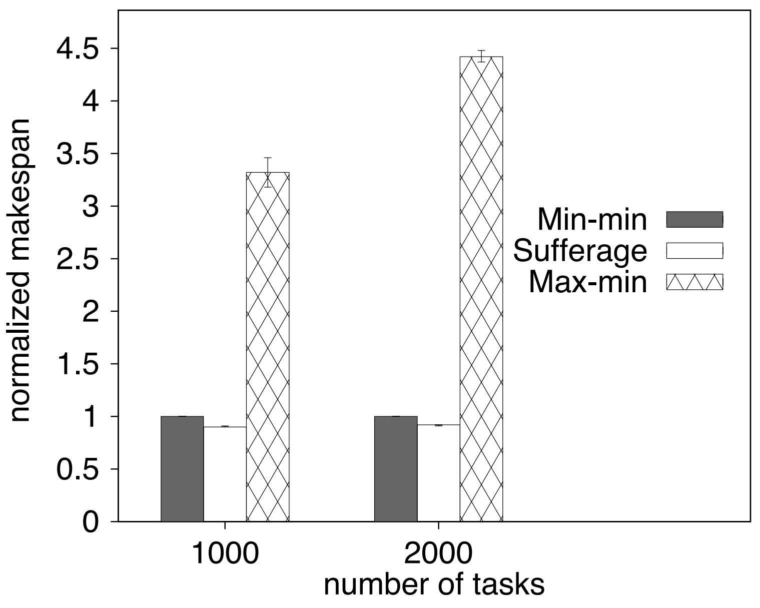

Fig. 6.11. Makespan of the batch mode heuristics for the regular time interval strategy and semi-consistent HiHi heterogeneity.

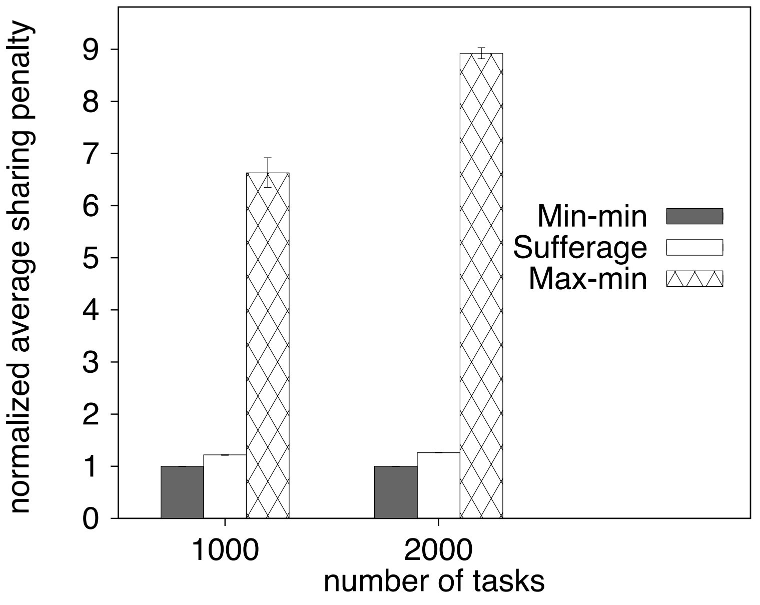

Fig. 6.12. Average sharing penalty of the batch mode heuristics for the regular time interval strategy and semi-consistent HiHi heterogeneity.

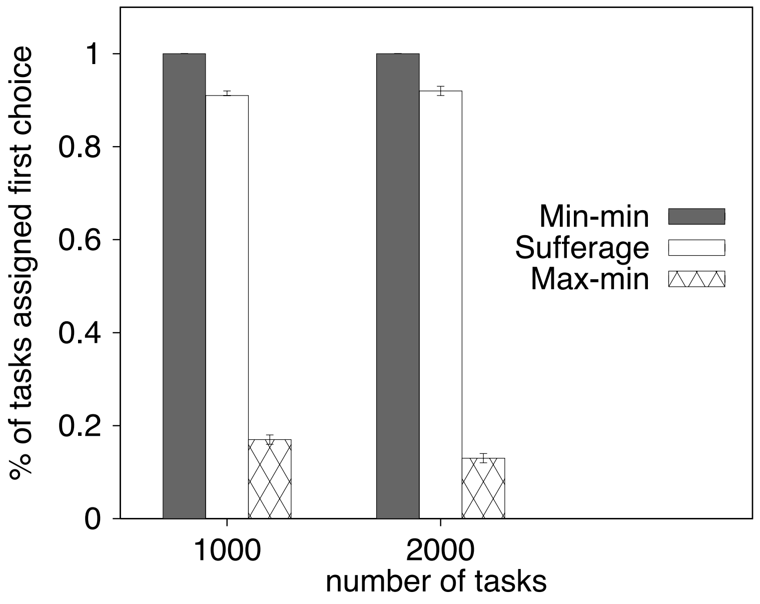

Fig. 6.13. PTBC for the batch mode heuristics for the regular time interval strategy and semi-consistent HiHi heterogeneity.

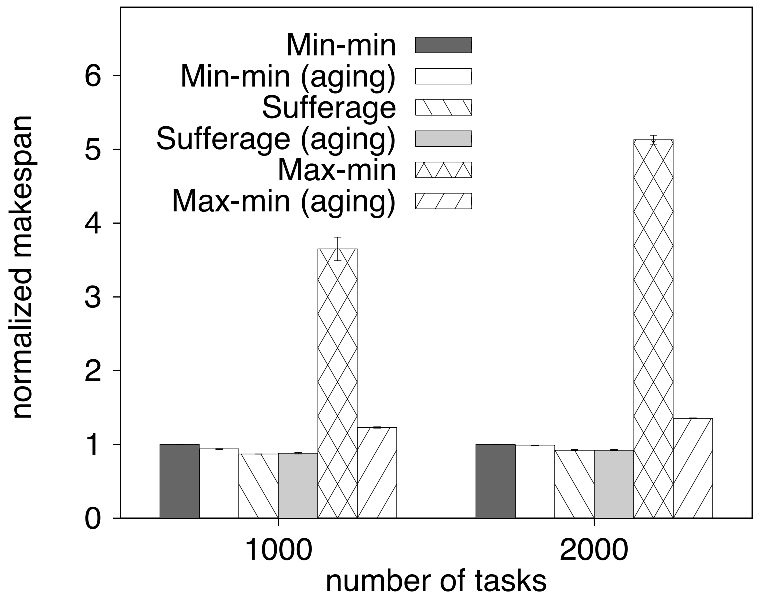

Fig. 6.14. Makespan for the batch mode heuristics for the regular time interval strategy with and without aging for inconsistent HiHi heterogeneity.

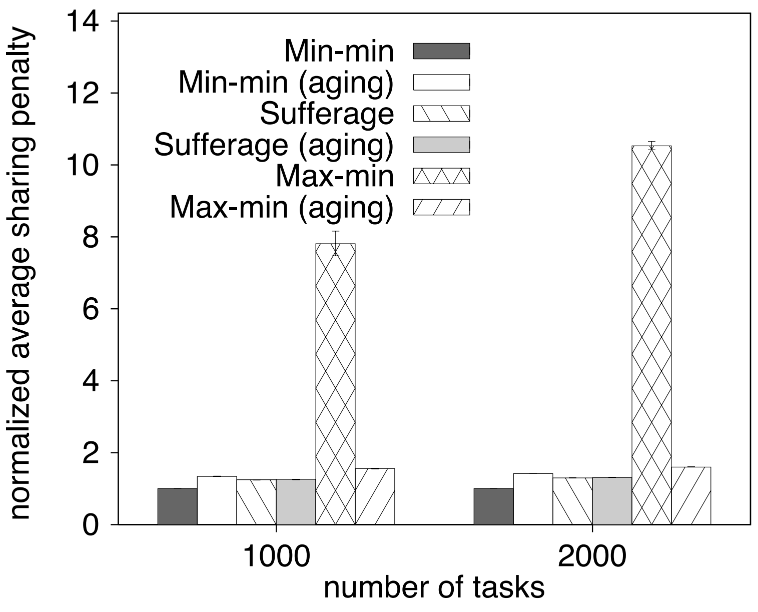

Fig. 6.15. Average sharing penalty of the batch mode heuristics for the regular time interval strategy with and without aging for inconsistent HiHi heterogeneity.

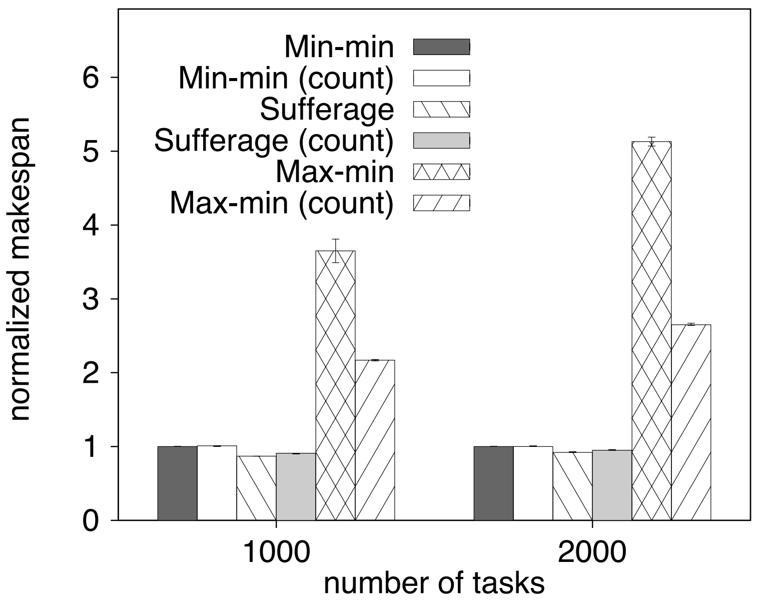

Fig. 6.16. Comparison of the makespans given by the regular time interval strategy and the Ţxed count strategy for inconsistent HiHi heterogeneity.

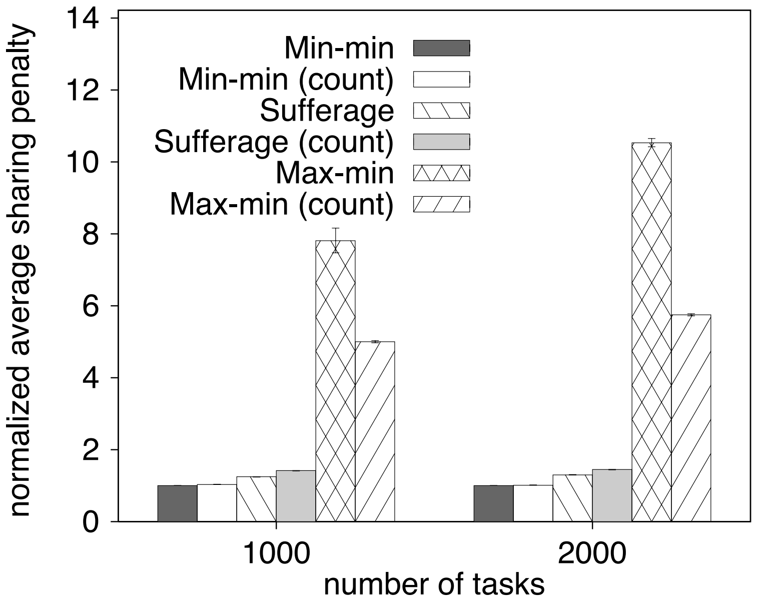

Fig. 6.17. Comparison of the average sharing penalty given by the Ţxed count mapping strategy and the regular time interval strategy for inconsistent HiHi heterogeneity.

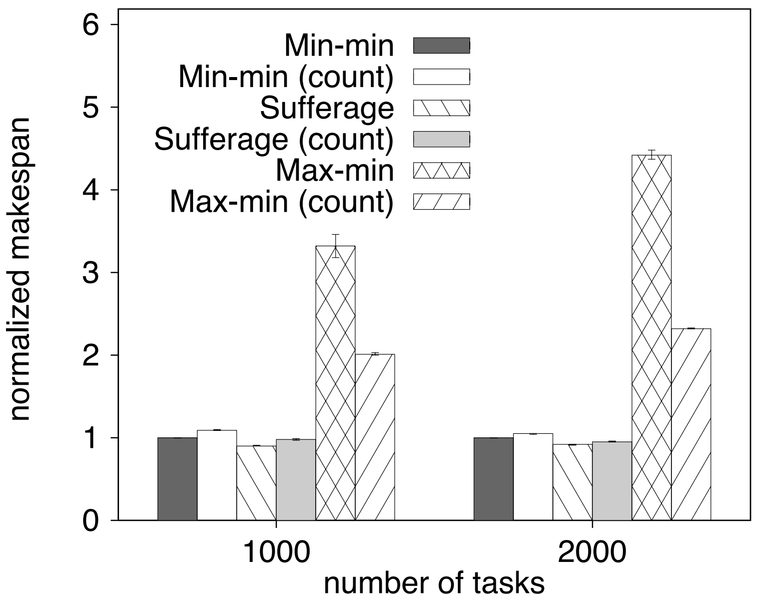

Fig. 6.18. Comparison of the makespan given by the Ţxed count mapping strategy and the regular time interval strategy for semi-consistent HiHi heterogeneity.

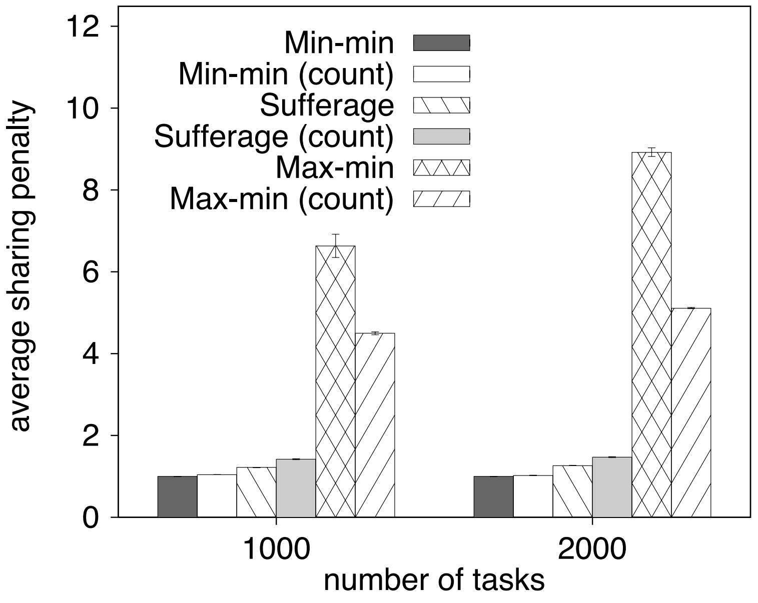

Fig. 6.19. Comparison of the average sharing penalty given by the Ţxed count mapping strategy and the regular time interval strategy for semi-consistent HiHi heterogeneity.

6.5.4 Comparing immediate and batch mode heuristics

In Figure 6.20, two immediate mode heuristics, the MCT and the KPB, are compared with two batch mode heuristics, the Min-min and the Sufferage. The comparison is performed with Poisson arrival rate set to λ_h. It can be noted that for this "high" arrival rate and $\mid K \mid$=2000, batch heuristics are superior to immediate mode heuristics. This is because the number of tasks waiting to begin execution is likely to be larger in the above circumstances than in any other considered here, which in turn means that rescheduling is likely to improve many more mappings in such a system. The immediate mode heuristics consider only one task when they try to optimize machine assignment, and do not reschedule. Recall that the mapping heuristics use a combination of expected and actual task execution times to compute machine ready times. The immediate mode heuristics are likely to approach the performance of the batch mode heuristics at low task arrival rates, because then both classes of heuristics will have comparable information about the actual execution times of the tasks. For example, at a certain low arrival rate, the 100-th arriving task might Ţnd that 70 previously arrived tasks have completed. At a higher arrival rate, only 20 tasks might have completed when the 100-th task arrived. The above observation is supported by the graph in Figure 6.21, which shows that the relative performance difference between immediate and batch mode heuristics decreases with a decrease in arrival rate. Given the observation that the KPB and the Sufferage perform almost similarly at this low arrival rate, it might be better to use the KPB heuristic because of its smaller computational complexity.

Figure 6.22 shows the performance difference between immediate and batch mode heuristics at an even faster arrival rate of λ_t. It can be seen that for $\mid K \mid$=2000 batch mode heuristics outperform immediate mode heuristics with a larger margin here. Although not shown in the results here, the makespan values for all heuristics are larger for lower arrival rate. This is attributable to the fact that at lower arrival rates, there is typically more machine idle time. Figures 6.23, 6.24, and 6.25 show the normalized average sharing penalty results for the three arrival rates discussed above. Once again the performance gap between batch and immediate heuristics increases with the increasing arrival rate, the reasons being the same as those given to explain relative makespan values.

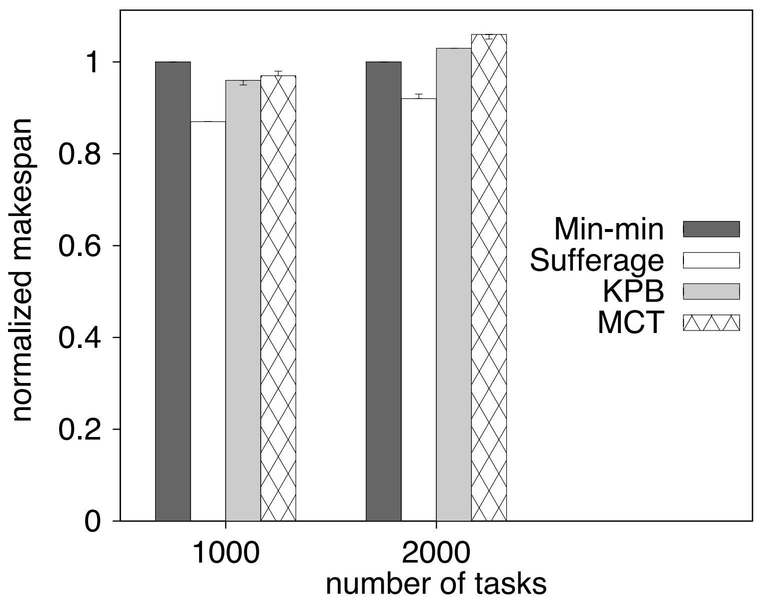

Fig. 6.20. Comparison of the makespan given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_h.

6.6 Summary

In the immediate mode, for both the semi-consistent and the inconsistent types of HiHi heterogeneity, the KPB heuristic outperformed the other heuristics on both performance metrics (however, the KPB was only slightly better than the MCT). The average sharing penalty gains were smaller than the makespan ones. The KPB can provide good system oriented performance (e.g., minimum makespan) and at the same time provide good application oriented performance (e.g., low average sharing penalty). The relative performance of the OLB and the MET with respect to the makespan reversed when the heterogeneity was changed from the inconsistent to the semi-consistent. The OLB did better than the MET for the semi-consistent case.

In the batch mode, for the semi-consistent and the inconsistent types of HiHi heterogeneity, the Min-min heuristic outperformed the Sufferage and Max-min heuristics in the average sharing penalty. However, the Sufferage performed the best with respect to

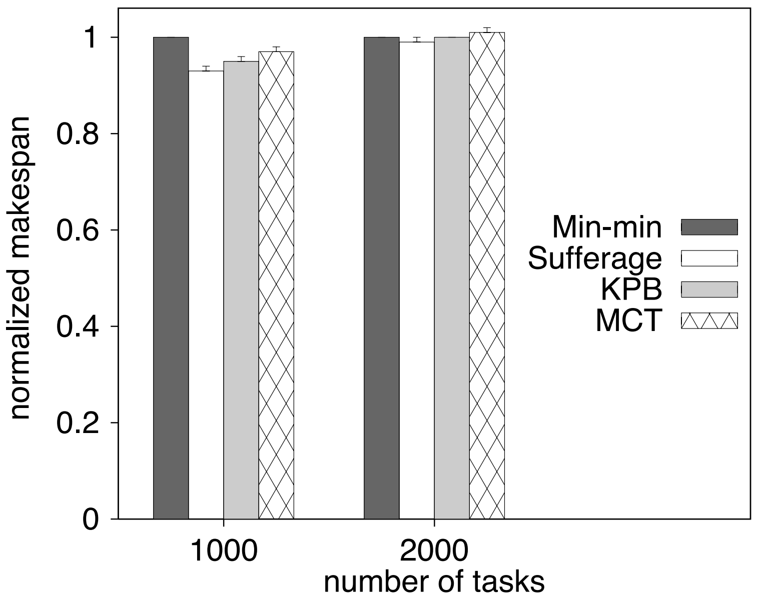

Fig. 6.21. Comparison of the makespan given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_l.

makespan (though, the Sufferage was only slightly better than the Min-min). The batch mode heuristics were shown to give a smaller makespan than the immediate ones for large $\mid K \mid$ and high task arrival rate. For smaller values of $\mid K \mid$ and lower task arrival rates, the improvement in makespan offered by batch mode heuristics was shown to be nominal.

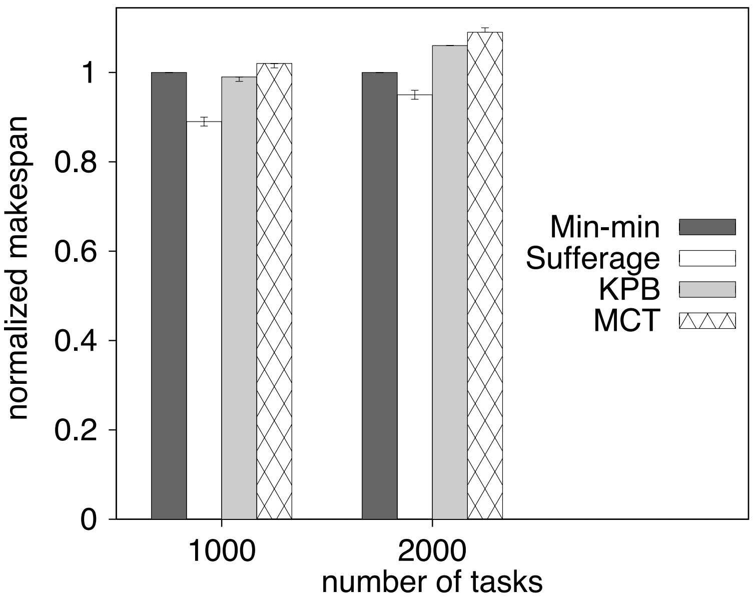

Fig. 6.22. Comparison of the makespan given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_t.

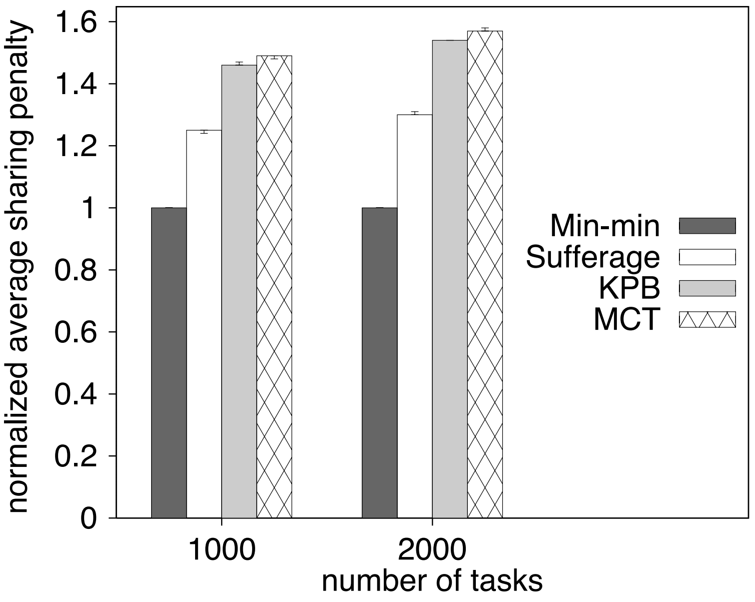

Fig. 6.23. Comparison of the average sharing penalty given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_h.

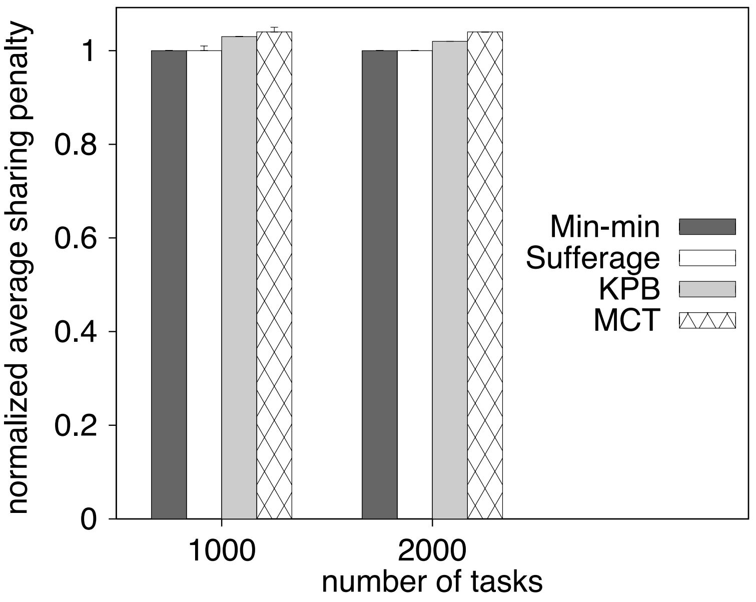

Fig. 6.24. Comparison of the average sharing penalty given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_l.

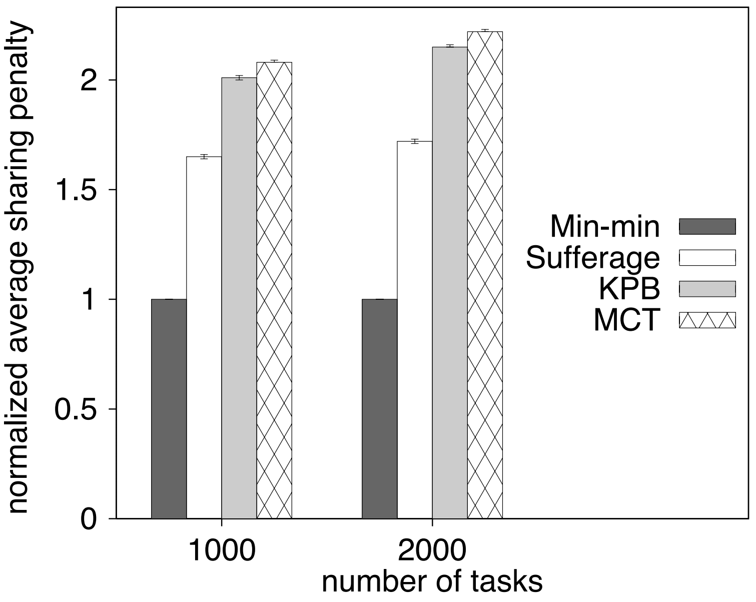

Fig. 6.25. Comparison of the average sharing penalty given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_t.

CHAPTER 7

RELATED WORK: STATIC HEURISTICS

7.1 Overview

This chapter describes and compares eleven static heuristics that can be used in an RMS like MSHN for mapping meta-tasks to machines. In a general HC system, static mapping schemes are likely to make better mapping decisions because more time can be devoted for the computation of schedules off-line than “immediately” in real-time. However, static schemes require that the set of tasks to be mapped be known *a priori*, and that the estimates of expected execution times of all tasks on all machines be known with reasonable accuracy. A meta-task, in the context of static heuristics, is the set of all independent tasks that are being considered for mapping. Like the dynamic heuristics in the previous chapter, these static mapping heuristics are non-preemptive, assume that the tasks have no deadlines or priorities associated with them, and assume a dedicated HC system.

7.2 Description of Static Heuristics

This section consists of brief deȚnitions of the eleven static meta-task mapping heuristics that are studied and fully described in [7]. The basic terms and the performance measure deȚned for the dynamic heuristics in Section 6.3 and Subsection 6.4.1 hold for static heuristics as well, except for the terms that characterize the dynamic nature of the dynamic heuristics, e.g., Țxed count strategy.

The descriptions below assume that the machine ready times are updated after each task is mapped. For cases when tasks can be considered in an arbitrary order, the order used is the one in which the tasks appeared in the ETC matrix.

This chapter has been summarized from a conference paper [7].

The static opportunistic load balancing (OLB) heuristic is similar to its dynamic counterpart except that it assigns tasks in an arbitrary order, instead of order of arrival. The user directed assignment (UDA) heuristic [1] works in the same way as the MET heuristic except that it maps tasks in an arbitrary order instead of order of arrival. The fast greedy heuristic [1] is the same as the MCT, except that it maps tasks in an arbitrary order instead of their order of arrival. The static Min-min heuristic works in the same way as the dynamic Min-min, except a meta-task contains all the tasks in the system. The static Max-min heuristic works in the same way as the dynamic Max-min, except a meta-task has all the tasks in the system. The greedy heuristic performs both the static Min-min and static Max-min heuristics, and uses the better solution [1, 19].

The genetic algorithm (GA) is a popular technique used for searching large solution spaces. The version of the heuristic used for this study was adapted from [55] for this particular HC environment. Figure 7.1 shows the steps in a general genetic algorithm [50].

```
(1)  initial population generation;
(2)  evaluation;
(3)  while (stopping criteria not met)
(4)       selection;
(5)       crossover;
(6)       mutation;
(7)       evaluation;
(8)  endwhile
```

Fig. 7.1. General procedure for a genetic algorithm.

The genetic algorithm implemented here operates on a population of 200 chromosomes (possible mappings) for a given meta-task. Each chromosome is a $| K |$ vector, where position i $(0 \leq i < t)$ is the machine to which the task t_i has been mapped. The initial population is generated using two methods: (a) 200 chromosomes randomly generated from a uniform distribution, or (b) one chromosome that is the Min-min solution and 199 random chromosomes. The latter method employs the seeding of the population with a Min-min chromosome. In this implementation, the GA executes eight times (four times with initial populations from each method), and the best of the eight mappings is used as the Ţnal solution. The makespan serves as the Ţtness value for evaluation of the evolution.

Simulated annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This solution uses the same representation for a solution as the chromosome for the GA. SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [45]). This probability is based on a system temperature that decreases for each iteration. As the system temperature "cools," it is more difŢcult for currently poorer solutions to be accepted.

The genetic simulated annealing (GSA) heuristic is a combination of the GA and SA techniques [46]. In general, GSA follows procedures similar to the GA outlined above. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simpliŢed SA decision process for accepting or rejecting new chromosomes.

The Tabu search keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these "Tabu" areas [23]. A solution (mapping) uses the same representation as a chromosome in the GA approach. Heuristic searches are conducted within a region, and the best solution for that region is stored. Then, a new region, not on the tabu list, is searched. When a stopping criterion is reached, the best solution among regions is selected.

The Ţnal heuristic in the comparison study is known as the A* heuristic. A* is a tree-based search that has been applied to many other task allocation problems (e.g., [10, 45]). The technique used here is similar to the one described in [10]. As the tree grows, intermediate nodes represent partial solutions (a subset of tasks are assigned to machines), and leaf nodes represent Ţnal solutions (all tasks are assigned to machines). The partial solution of a child node has one more task t_a mapped than the parent node. Each parent node can be replaced by its m children, one for each possible mapping of t_a. The number of nodes allowed in the tree is bounded to limit mapper execution time. Less promising nodes are deleted, and the more promising nodes are expanded. The process continues until a leaf node (complete mapping) is reached.

7.3 Sample Comparisons for Static Mapping Heuristics

Figures 7.2 and 7.3 show comparisons of the eleven static heuristics using makespan as the criterion in two different heterogeneity environments. Vertical lines at the top of the bars show minimum and maximum values for the 100 trials, while the bars show the averages. It can be seen that, for the parameters used in this study, GA gives the smallest makespan for both inconsistent HiHi and inconsistent HiLo heterogeneities. The reader is referred to [7] for more results, details, and discussions.

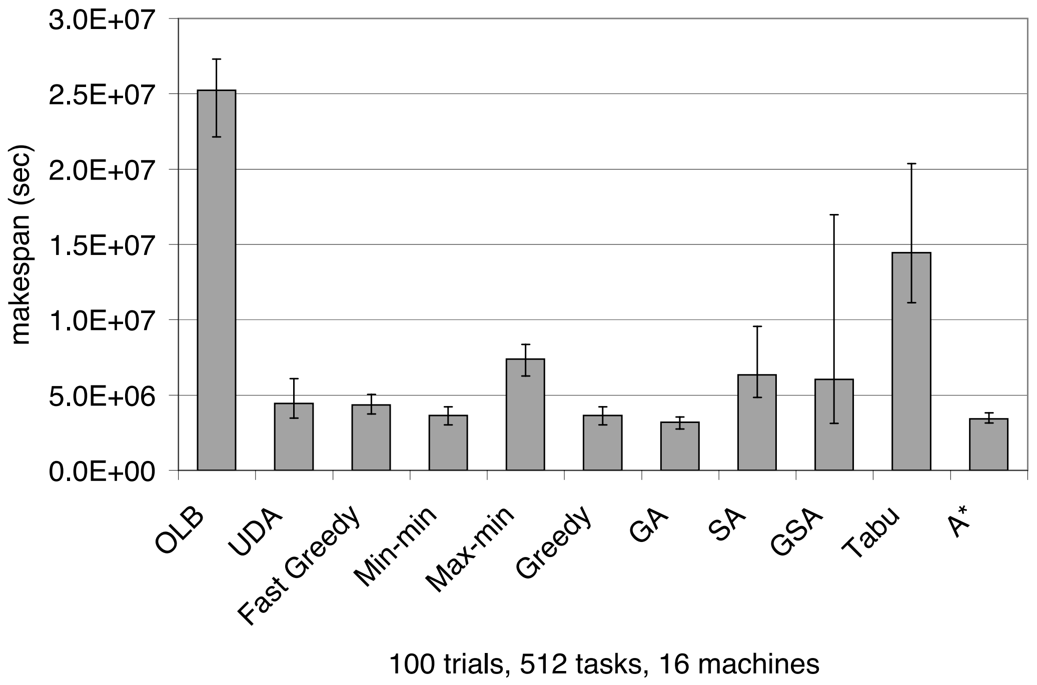

Fig. 7.2. Comparison of the makespan given by sample static heuristics for inconsistent HiHi heterogeneity.

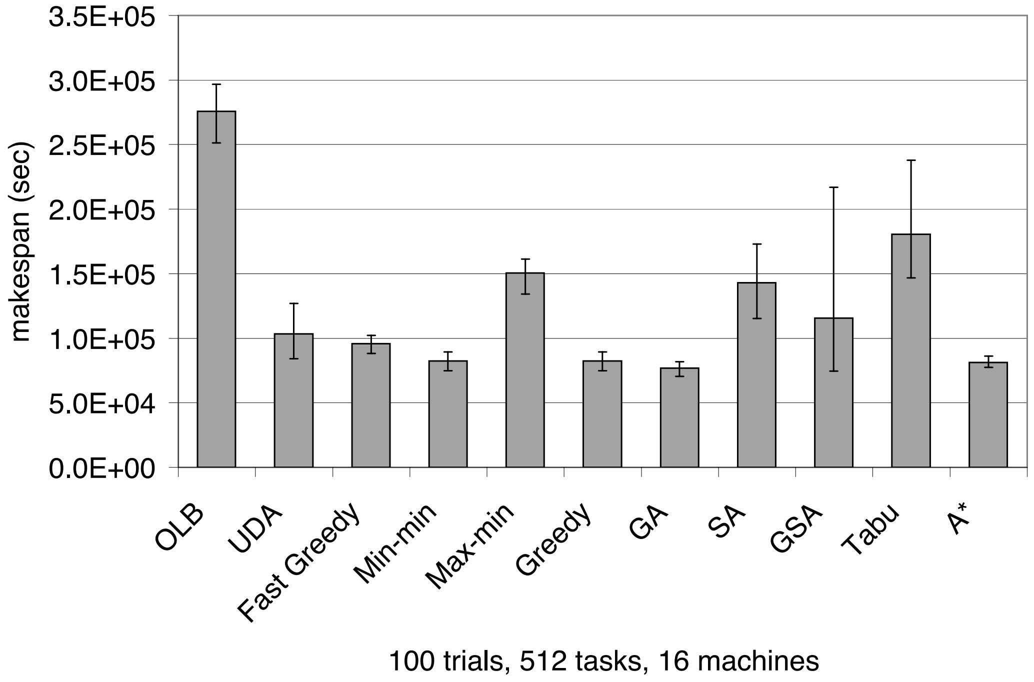

Fig. 7.3. Comparison of the makespan given by sample static heuristics for inconsistent HiLo heterogeneity.

CHAPTER 8

CONCLUSIONS

Heterogeneous computing is a relatively new research area for the computer Ţeld. Interest in such systems continues to grow, both in the research community and in the user community.

Some of the different types of HC systems that have been built were discussed here, including mixed-mode, multi-mode, and mixed-machine. Mixed-machine HC was then focussed upon. As an example of the design of a resource management system for such HC environments, the high-level functional architecture of MSHN was provided.

To describe different kinds of heterogeneous environments, a model based on characteristics of the ETC matrix was presented. The three parameters of this model (task heterogeneity, machine heterogeneity, and consistency) can be changed to investigate the performance of mapping heuristics for different HC systems and different sets of tasks. Heterogeneity was quantiŢed in two different ways. Corresponding procedures for generating the ETC matrices representing various heterogeneous environments were presented. Several sample ETC matrices were provided for the more involved ETC generation procedure.

The mapping of tasks and meta-tasks, and the scheduling of communications, in HC environments are active, growing areas of research. Based on existing mapping approaches in the literature, a three-part taxonomy was proposed. The Purdue HC Taxonomy classiŢed those characteristics of applications, target platforms, and the mapping strategy that affect the quality of mapping. These deŢning traits are incorporated in taxonomy as application model, target platform model, and mapping strategy model. By deŢning these three models, heterogeneous mapping techniques can be classiŢed more accurately. A sample listing of mapping heuristics with their Purdue HC Taxonomy classiŢcation was given to illustrate application of the taxonomy.

Dynamic matching and scheduling heuristics for mapping independent tasks onto HC systems were compared under a variety of simulated computational environments. Five on-line mode heuristics and three batch mode heuristics were studied.

This study quantiŢed how the relative performance of these dynamic mapping heuristics depends on (a) the consistency property of the ETC matrix, (b) the requirement to optimize system oriented or application oriented performance metrics (e.g., optimizing makespan versus optimizing average sharing penalty), and (c) the arrival rate of the tasks. Thus, the choice of the heuristic that is best to use in a given heterogeneous environment will be a function of such factors. Therefore, it is important to include a set of heuristics in a resource management system for HC, and then use the heuristic that is most appropriate for a given situation (as will be done in the Scheduling Advisor for MSHN).

Researchers can build on the evaluation techniques and results presented here in future efforts by considering other non-preemptive dynamic heuristics, as well as preemptive ones. Furthermore, in future studies, tasks can be characterized in more complex ways (e.g., inter-task communications, deadlines, priorities [6]) and using other environmental factors (e.g., task arrival rates, degrees of heterogeneity, number of machines in the HC suite, impact of changing the variance when simulating actual task execution times).

Thus, this thesis gives some techniques to model heterogeneity, delineates a taxonomy for characterizing mapping strategies, examines important heuristics, and provides comparisons, as well as acts as a framework for future research.

LIST OF REFERENCES

[1] R. Armstrong, D. Hensgen, and T. Kidd, “The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions,” in *7th IEEE Heterogeneous Computing Workshop (HCW ’98)*, Mar. 1998, pp. 79–87.

[2] R. Armstrong, “Investigation of Effect of Different Run-Time Distributions on SmartNet Performance,” Master’s thesis, Department of Computer Science, Naval Postgraduate School, 1997 (D. Hensgen, Advisor).

[3] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, “Adaptive communication algorithms for distributed heterogeneous systems,” in *IEEE International Symposium on High Performance Distributed Computing*, July 1998, pp. 310–321.

[4] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, “Block-cyclic redistribution over heterogeneous networks,” in *International Conference on Parallel and Distributed Computing Systems*, Sep. 1998, pp. 242–249.

[5] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, “EfŢcient collective communication in distributed heterogeneous systems,” in *IEEE International Conference on Distributed Computing Systems*, June 1999, pp. 15–24.

[6] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, “A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems,” in *1998 IEEE Symposium on Reliable Distributed Systems*, Oct. 1998, pp. 330–335.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, R. F. Freund, and D. Hensgen, “A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems,” in *8th IEEE Heterogeneous Computing Workshop (HCW ’99)*, Apr. 1999, pp. 15-29.

[8] A. H. Buss, “A tutorial on discrete-event modeling with simulation graphs,” in *1995 Winter Simulation Conference (WSC ’95)*, Dec. 1995, pp. 74–81.

[9] T. L. Casavant and J. G. Kuhl, “A taxonomy of scheduling in general-purpose distributed computing systems,” *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, Feb. 1988, pp. 141–154.

[10] K. Chow and B. Liu, “On mapping signal processing algorithms to a heterogeneous multiprocessor system,” in *1991 International Conference on Acoustics, Speech, and Signal Processing - ICASSP 91*, Vol. 3, 1991, pp. 1585–1588.

[11] J. Duato, S. Yalmanchili, and L. Ni, “Interconnection Networks: An Engineering Approach,” IEEE Computer Society Press, Los Alamitos, CA, 1997.

[12] I. Ekmečić, I. Tartalja, and V. Milutinović, “A taxonomy of heterogeneous computing,” *IEEE Computer*, Vol. 28, No. 12, Dec. 1995, pp. 68–70.

[13] I. Ekmečić, I. Tartalja, and V. Milutinović, “A survey of heterogeneous computing: Concepts and systems,” *Proceedings of the IEEE*, Vol. 84, No. 8, Aug. 1996, pp. 1127–1144.

[14] M. M. Eshaghian and Y.-C. Wu, “A portable programming model for network heterogeneous computing,” *in* “Heterogeneous Computing” (M. M. Eshaghian, Ed.), Artech House, Norwood, MA, 1996, pp. 155–195.

[15] M. M. Eshaghian (ed.), “Heterogeneous Computing,” Artech House, Norwood, MA, 1996.

[16] D. Fernandez-Baca, “Allocating modules to processors in a distributed system,” *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[17] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, “Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting,” *Journal of Parallel and Distributed Computing*, Vol. 11, No. 3, Mar. 1991, pp. 239–251.

[18] I. Foster and C. Kesselman (eds.), “The Grid: Blueprint for a New Computing Infrastructure,” Morgan Kaufmann, San Fransisco, CA, 1999.

[19] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, “Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet,” in *7th IEEE Heterogeneous Computing Workshop (HCW ’98)*, Mar. 1998, pp. 184–199.

[20] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, “SmartNet: A scheduling framework for metacomputing,” in *2nd International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN ’96)*, June 1996, pp. 514–521.

[21] R. F. Freund and H. J. Siegel, “Heterogeneous processing,” *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13–17.

[22] A. Ghafoor and J. Yang, “Distributed heterogeneous supercomputing management system,” *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[23] F. Glover and M. Laguna, “Tabu Search,” Kluwer Academic Publishers, Boston, MA, 1997.

[24] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, “An overview of MSHN: The Management System for Heterogeneous Networks,” in *8th IEEE Heterogeneous Computing Workshop (HCW ’99)*, Apr. 1999, pp. 184–198.

[25] O. H. Ibarra and C. E. Kim, “Heuristic algorithms for scheduling independent tasks on nonidentical processors,” *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[26] M. A. Iverson and F. Ozguner, “Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment,” in *7th IEEE Heterogeneous Computing Workshop (HCW ’98)*, Mar. 1998, pp. 70–78.

[27] R. Jain, “The Art of Computer Systems Performance Analysis,” John Wiley & Sons, Inc., New York, NY, 1991.

[28] M. KaṬl and I. Ahmad, “Optimal task assignment in heterogeneous computing systems,” in *6th IEEE Heterogeneous Computing Workshop (HCW ’97)*, Apr. 1997, pp. 135–146.

[29] M. KaṬl and I. Ahmad, “Optimal task assignment in heterogeneous distributed computing systems,” *IEEE Concurrency*, Vol. 6, No. 3, July-Sep. 1998, pp. 42–51.

[30] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, “Heterogeneous computing: Challenges and opportunities,” *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18–27.

[31] J.-K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. St. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, “A Multi-Dimensional QoS Performance Measure for Distributed Heterogeneous Networks,”, Technical Report, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in preparation, 1999.

[32] C. Leangsuksun, J. Potter, and S. Scott, “Dynamic task mapping algorithms for a distributed heterogeneous computing environment,” in *4th IEEE Heterogeneous Computing Workshop (HCW ’95)*, Apr. 1995, pp. 30–34.

[33] Y. A. Li and J. K. Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," in *6th IEEE Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 172–184.

[34] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Special Issue on Software Support for Distributed Computing, Sep. 1999, to appear.

[35] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund, "A Comparison of Dynamic Strategies for Mapping a Class of Independent Tasks onto Heterogeneous Computing Systems,", Technical Report, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 1999, in preparation.

[36] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *in* "Encyclopedia of Electrical and Electronics Engineering" (J. G. Webster, Ed.), John Wiley, New York, NY, 1999, Vol. 8, pp. 679–690.

[37] M. Maheswaran and H. J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 57–69.

[38] M. Maheswaran, "Software Issues on Mapping Applications onto Heterogeneous Machines and the Performance of Krylov Algorithms on Parallel Machines," PhD thesis, School of Electrical and Computer Engineering, Purdue University, 1998.

[39] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Adaptive load sharing in heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, Vol. 9, No. 4, Aug. 1990, pp. 331–346.

[40] A. Papoulis, "Probability, Random Variables, and Stochastic Processes," McGraw-Hill, New York, NY, 1984.

[41] M. Pinedo, "Scheduling: Theory, Algorithms, and Systems," Prentice Hall, Englewood Cliffs, NJ, 1995.

[42] U. W. Pooch and J. A. Wall, "Discrete Event Simulation: A Practical Approach," CRC Press, Boca Raton, FL, 1993.

[43] R. van Renesse and A. S. Tanenbaum, "Distributed operating systems," *ACM Computing Surveys*, Vol. 17, No. 4, Dec. 1985, pp. 419–470.

[44] H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems," *IEE Proceedings on Computer and Digital Techniques*, Vol. 141, No. 1, Jan. 1994, pp. 1–10.

[45] S. J. Russell and P. Norvig, “ArtiŢcial Intelligence: A Modern Approach,” Prentice Hall, Englewood Cliffs, NJ, 1995.

[46] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, “Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments,” in *5th IEEE Heterogeneous Computing Workshop (HCW ’96)*, Apr. 1996, pp. 98–117.

[47] H. J. Siegel, H. G. Dietz, and J. K. Antonio, “Software support for heterogeneous computing,” *in* “The Computer Science and Engineering Handbook” (J. A. B. Tucker, Ed.), CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.

[48] H. J. Siegel, M. Maheswaran, D. W. Watson, J. K. Antonio, and M. J. Atallah, “Mixed-mode system heterogeneous computing,” *in* “Heterogeneous Computing” (M. M. Eshaghian, Ed.), Artech House, Norwood, MA, 1996, pp. 19–65.

[49] H. Singh and A. Youssef, “Mapping and scheduling heterogeneous task graphs using genetic algorithms,” in *5th IEEE Heterogeneous Computing Workshop (HCW ’96)*, Apr. 1996, pp. 86–97.

[50] M. Srinivas and L. M. Patnaik, “Genetic algorithms: A survey,” *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17–26.

[51] V. Suresh and D. Chaudhuri, “Dynamic rescheduling–A survey of research,” *International Journal of Production Economics*, Vol. 32, No. 1, Aug. 1993, pp. 53–63.

[52] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, “Minimizing the application execution time through scheduling of subtasks and communication trafŢc in a heterogeneous computing system,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, Aug. 1997, pp. 857–871.

[53] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk, “A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment,” in *7th IEEE Heterogeneous Computing Workshop (HCW ’98)*, Mar. 1998, pp. 115–129.

[54] P. Tang, P. C. Yew, and C. Zhu, “Impact of self-scheduling on performance of multiprocessor systems,” in *3rd International Conference on Supercomputing*, July 1988, pp. 593–603.

[55] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, “Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach,” *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 8–22.

[56] C. C. Weems, S. Levitan, A. R. Hanson, E. M. Riseman, and J. G. Nash, “The image understanding architecture,” *International Journal of Computer Vision*, Vol. 2, No. 3, Jan. 1989, pp. 251–282.

APPENDIX

SOURCE CODE

This appendix gives the source code used in the simulations. A makefile has been provided to compile the source code. The simulator is invoked by the command `msimu`. Some of the command line options specify the HC environment, the heuristic to be evaluated, and the task arrival rate. Complete option summary could be found by typing just `msimu`.